Genomics and Bioinformatics: Week 4

8 October 2013

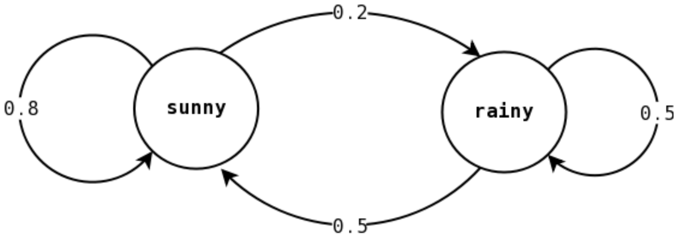
1 Markov Models

In mathematics Markov(ian) models are a way of describing "memory-less" time evolutions. We assume that there is a finite number of possible situations (the *state space* S) and as time advances by discrete steps (t=$1, 2, 3, 4, \ldots, n$), the system transits from one state to another. The probability of going into a new state k depends only on the current state j, hence the absence of memory from previous events. As an example, we may have two states describing the weather condition, $S = \{\text{sunny}, \text{rainy}\}$, and the following probabilities:

$$M = \begin{array}{c|cc} & \text{sunny} & \text{rainy} \\ \hline \text{sunny} & 0.8 & 0.5 \\ \text{rainy} & 0.2 & 0.5 \end{array} .$$

where $M(j, k)$ is the probability of transiting from state j at time t to state k at the next time $t + 1$.

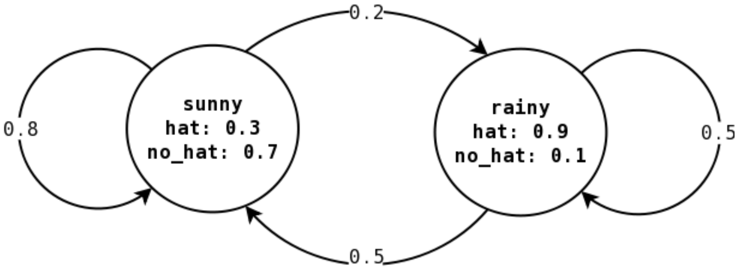
We often represent these models graphically as follows:



Remark that we can assign a probability to each state (*e.g.*, $\pi_{\text{sunny}}(t) = 0.1$, $\pi_{\text{rainy}}(t) = 0.9$) and multiplying the vector $\pi(t) = (\pi_{\text{sunny}}(t), \pi_{\text{rainy}}(t))$ by M we get a new vector of probabilities at $t + 1$: $\pi(t + 1) = (0.53, 0.47)$

2 Hidden States

Assume now that we can observe a certain sequence of events which are triggered by underlying unobservable states of the system. For example suppose that your neighbour will wear a hat 9 times out of 10 on rainy days and 3 times out of 10 on sunny days (symbolize this by the function E: $E(\text{hat}, \text{rainy}) = 0.9$, $E(\text{hat}, \text{sunny}) = 0.3$). By observing your neighbour every day, you can try to guess the weather without actually looking at it. We model this by extending Markov models as follows: each (now hidden) state can *emit* symbols from a given alphabet, with a certain probability distribution:



Such a model will randomly generate two sequences: $\mathcal{S}$, for the hidden states (`S`: sunny, `R`: rainy) and $\mathcal{T}$, for the observed symbols (`H`: hat, `N`: no hat), like for example:

```
HHNHHHHNNNHHNHHNHHNHNNNHHNNNNNHNHNNNNHNNH
SSSSRRSSSSRSSSRSRSSSSSSSRRRSSSSSSRSSSSSSR
```

There are three main questions we are going to address on such models:

1. What is the sequence $\mathcal{S}$ of hidden states which is most likely to have generated the observed symbols $\mathcal{T}$ (Viterbi algorithm),

2. What is the probability of an observed sequence $\mathcal{T}$ of symbols (Forward/Backward algorithms),

3. What are the parameters E, M (emission and transition probabilities) that maximize the probability of the observed sequence $\mathcal{T}$ (Baum-Welch algorithm).

Let us next write a few formulas that will allow us to address the above questions. Using the transition probabilities, we can compute the total probability of the hidden state sequence:

$$P(\mathcal{S}) \;=\; P(\mathcal{S}_0) \prod_{n=1}^{N} M(\mathcal{S}_{n-1}, \mathcal{S}_n) \;,$$

(we have introduced an additional hidden state $\mathcal{S}_0$ with a certain probability $P(\mathcal{S}_0)$ to be determined). Similarly we can compute the probability of the emitted symbols sequence knowing the hidden states:

$$P(\mathcal{T}|\mathcal{S}) \;=\; \prod_{n=1}^{N} P(\mathcal{T}_n|\mathcal{S}_n) \;=\; \prod_{n=1}^{N} E(\mathcal{T}_n, \mathcal{S}_n) \;.$$

We can now use Bayes' Theorem to infer the probability of the hidden sequence from the observed symbols:

$$P(\mathcal{S}|\mathcal{T}) \;=\; \frac{P(\mathcal{T}|\mathcal{S})P(\mathcal{S})}{P(\mathcal{T})} \;=\; \frac{P(\mathcal{T}|\mathcal{S})P(\mathcal{S})}{\sum_\sigma P(\mathcal{T}|\sigma)P(\sigma)}\ .$$

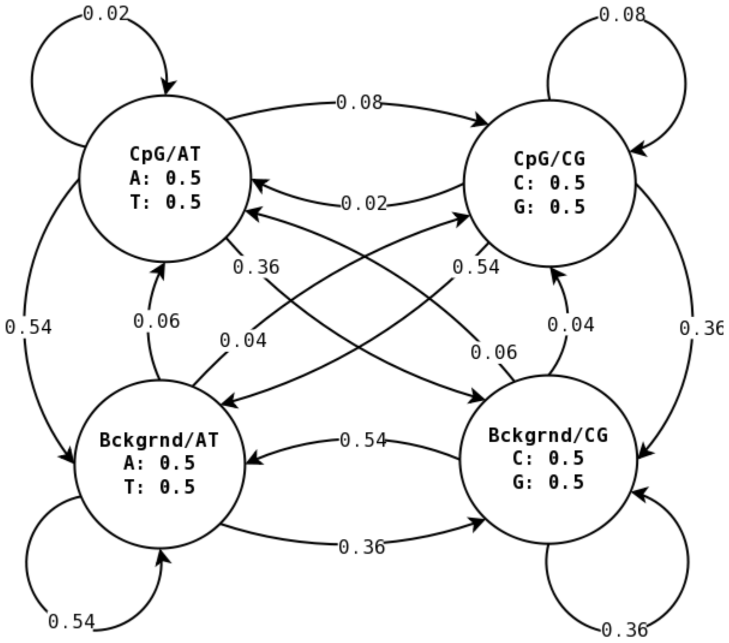
The most likely sequence of hidden states (our guess for the weather) will be the sequence $\mathcal{S}$ maximizing this probability (called *the posterior*).

There are two technical points that we will mention here: first it is generally dangerous to compute long products of small numbers because it will rapidly *underflow* (become effectively 0 for the computer). We therefore always compute in terms of the log of the probabilities:

$$\log P(\mathcal{S}) \;=\; \log P(\mathcal{S}_0) + \sum_{n=1}^{N} \log M(\mathcal{S}_{n-1}, \mathcal{S}_n)\ ,$$

Secondly, it is often awkward to score the first and the last character in the sequence, and we therefore sometimes introduce *begin* and *end* states. or equivalently initial and final state probability distributions. For example we can specify $P(\mathcal{S}_0)$ for each $\mathcal{S}_0$ or say that the first state is always *begin* (which doesn't emit anything) and specify transitions $M(\text{begin}, \mathcal{S}_1)$. Similarly for the last or end state. A typical choice is a begin state with equal probability of transition to every other state to avoid introducing a bias.

3 Example 1: CpG islands



This diagram shows a simple application of HMMs to a biological problem: identifying CpG islands in a genome. This example is rather untypical because the emission probabilities are polarized (many symbols have zero emission probability). The basic fact that we will use is that the frequency of C-G step is extremely low in the genome (about 1%) except within CpG islands.

4 The Viterbi Algorithm

The Viterbi algorithm is an efficient way of finding the maximum likelihood path in the HMM corresponding to an observed sequence of symbols. It is another example of dynamic programming, very similar to the Needleman-Wunsch algorithm. The scoring function is

$$F_{n,s} \;=\; E(T_n, s) \max_{\sigma} F_{n-1,\sigma} M(\sigma, s) \; .$$

Then backtracking proceeds as usual from the final state with highest score at step n following the arrows that produced the maximum at each step.

5 The Forward and Backward Algorithms

The Viterbi algorithm provides the best-scoring path, but to know the actual probability of the data (given the model), we should sum over all possible paths:

$$P(\mathcal{T}) \;=\; \sum_{\mathcal{S}} P(\mathcal{T}|\mathcal{S}) P(\mathcal{S}) \; .$$

There is a simple modification of the Viterbi algorithm called the *forward* algorithm which amounts to replacing the max by a sum:

$$F_{n,s} \;=\; \sum_{\sigma} M(\sigma, s) E(T_n, s) F_{n-1,\sigma} \; .$$

By construction, the score $F_{n.s}$ represents the probability of the observed sequence up to step n:

$$F_{n,s} \;=\; P(\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{S}_n = s) \; .$$

Therefore the final score in the *end* state is the probability of the observed data.

There is a similar algorithm, the *backward* algorithm which works backwards in the sequence of observed symbols:

$$B_{n,s} \;=\; \sum_{\sigma} M(s, \sigma) E(T_{n+1}, \sigma) B_{n+1,\sigma} \; ,$$

from the initial value $B_{N,\sigma} = 1$ for all σ. Similarly as the forward algorithm, these scores can be understood as the probability of the sequence of observed symbols from step $n + 1$ to the end of the sequence:

$$B_{n,s} \;=\; P(\mathcal{T}_{n+1}, \ldots, \mathcal{T}_N | \mathcal{S}_n = s) \; .$$

As a consequence we now have the probability of each state at step n jointly with the whole sequence of observed symbols given by:

$$\frac{F_{n,s} \cdot B_{n,s}}{F_{N,end}} = \frac{P(\mathcal{T}_1, \dots, \mathcal{T}_n, \mathcal{S}_n = s) \cdot P(\mathcal{T}_{n+1}, \dots, \mathcal{T}_N | \mathcal{S}_n = s)}{P(\mathcal{T})} = P(\mathcal{S}_n = s | \mathcal{T}) \ .$$

6 The Baum-Welch Algorithm

All the above algorithms were assuming a fixed HMM structure with known transition and emission rates. We will now discuss how these rates can be chosen or optimized. For this we will assume that we have a set of *training sequences*, that will be used to define the model which can then be applied to a new set of *test sequences*. First we assume that both the observed sequences $\mathcal{T}_1, \dots, \mathcal{T}_M$ and the corresponding hidden states $\mathcal{S}_1, \dots, \mathcal{S}_M$ are known. With them we can count the number of emissions from each state: $e(t, s)$ and the number of transitions between every pairs of states: $m(s, s')$. Then the best guess for the model parameters will be

$$\begin{aligned} E(t,s) &= \frac{e(t,s)}{\sum_\tau e(\tau, s)} \ , \\ M(s,s') &= \frac{m(s,s')}{\sum_\sigma m(s,\sigma)} \ . \end{aligned}$$

If the hidden states are not known, we can apply the iterative *Baum-Welch* algorithm: start from random values (or best guesses) of the parameters, then run the forward and backward algorithms to compute

$$\begin{aligned} e(t,s) &= \sum_{\mathcal{T} \in \text{training set}} \sum_{i:\mathcal{T}_i = t} P(\mathcal{S}_i = s | \mathcal{T}) \\ &= \sum_{\mathcal{T} \in \text{training set}} \frac{1}{P(\mathcal{T})} \sum_{i:\mathcal{T}_i = t} F_{i,s}(\mathcal{T}) B_{i,s}(\mathcal{T}) \ , \\ m(s,s') &= \sum_{\mathcal{T} \in \text{training set}} \sum_{i=1}^{N} P(\mathcal{S}_i = s, \mathcal{S}_{i+1} = s' | \mathcal{T}) \\ &= \sum_{\mathcal{T} \in \text{training set}} \frac{1}{P(\mathcal{T})} \sum_{i=1}^{N} F_{i,s}(\mathcal{T}) M(s,s') E(\mathcal{T}_{i+1}, s') B_{i+1,s'}(\mathcal{T}) \ , \end{aligned}$$

We plug these values into the formulas above to generate new sets of parameters and repeat the procedure until we decide that the parameters have reached stable values.

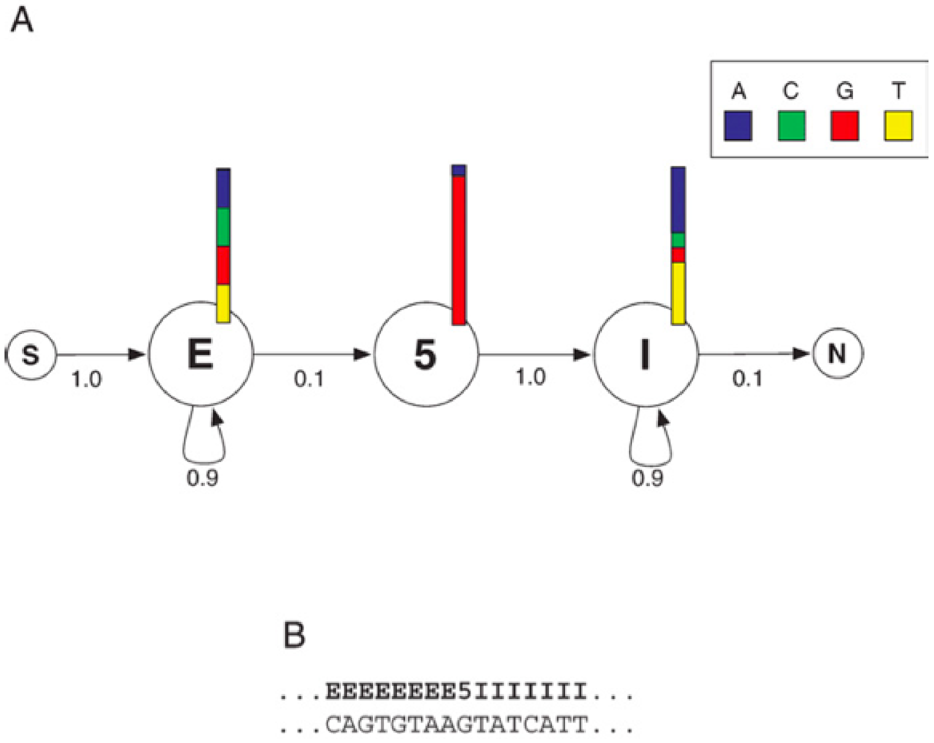
7 Example 2: Spliced genes

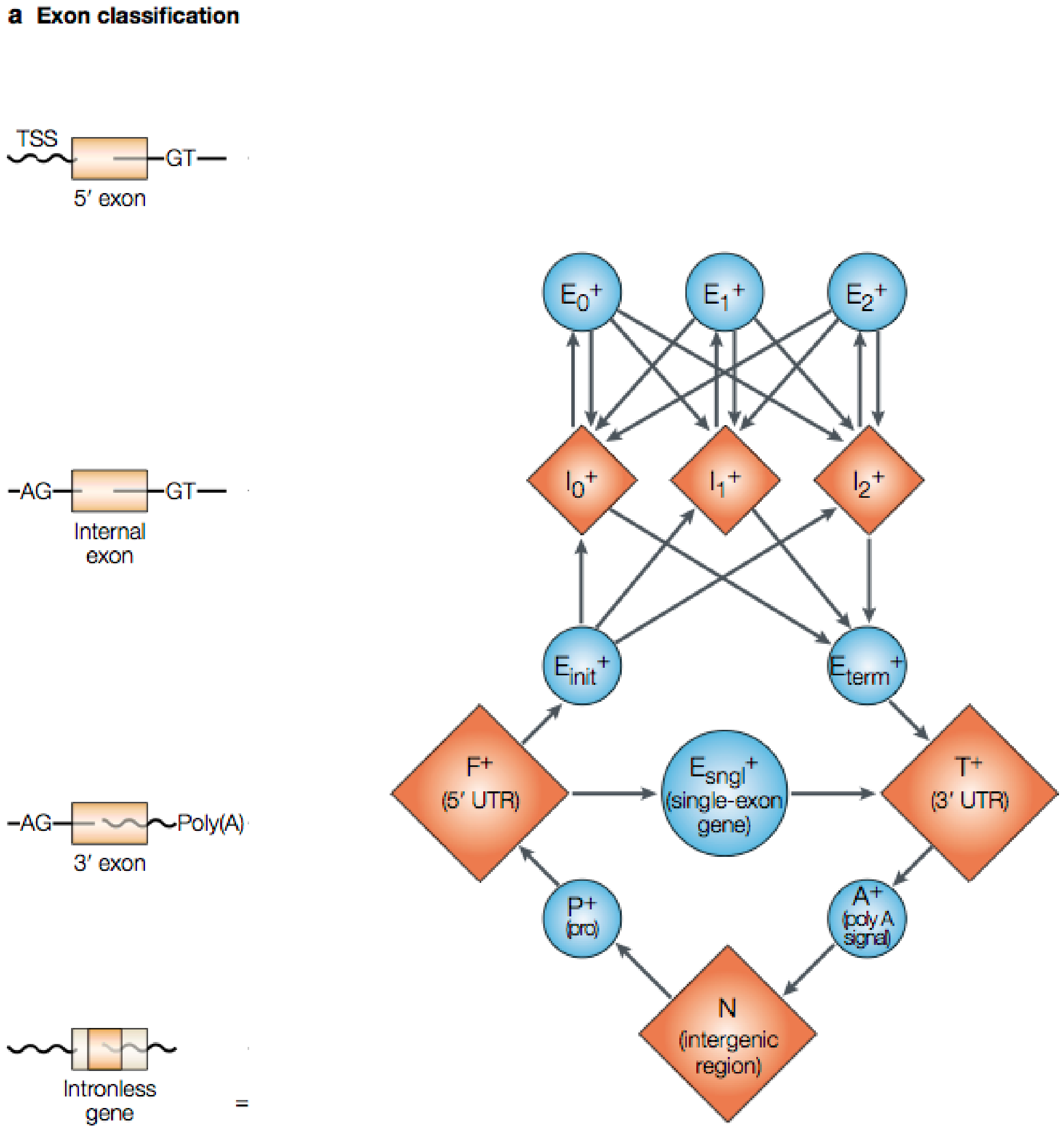
We finally present schematically a model used to identify spliced genes in a genome. The general idea is to collect observations about the structure of gene promoters, 3' and 5'UTRs and splice junctions in terms of sequence composition biases, and represent these as disctinct HMM states. Then use a set of well-annotated genes to train (with Baum-Welch) the parameters of the model, and finally to infer most likely hidden state sequences on a set of unannotated genomic sequences.

References

1. Schuster-Böckler, B., & Bateman, A. *An introduction to hidden Markov models*, Current protocols in bioinformatics, (2007).

2. Zhang, M.Q. *Computational prediction of eukaryotic protein-coding genes*, Nature Reviews Genetics, **3**, 698–709 (2002)





a Exon classification