Computational Biology
Lecture 10: Forward and backward evaluation, HMM training
Saad Mneimneh

We have seen in the previous lecture how to obtain the most probable path for a sequence $x_1...x_n$ using the Viterbi decoding algorithm. We now move the the question of evaluation: Given a sequence x, what is $p(x)$? We know how to compute $p(x,\pi)$ for a given π, but our difficulty now lies in the fact that we do not know the path π that generated x; therefore, we have to consider all of them and compute $p(x) = \sum_\pi p(x,\pi)$. Of course finding $p(x,\pi)$ for every possible π is not efficient since we have an exponential number of paths.

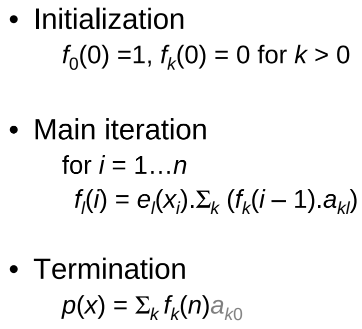
Forward evaluation

Luckily, $p(x)$ can itself be calculated using a similar dynamic programming procedure to the Viterbi algorithm, replacing the maximization steps with sums. This will be called the forward algorithm. Let $f_l(i)$ to be the probability $p(x_1...x_i, \pi_i = l)$. Then we can define $f_l(i)$ as

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

This is simply saying that the probability of obtaining $x_1...x_i$ and ending in state l is the probability of obtaining $x_1...x_{i-1}$ and ending in any state k, then making a transition to state l, and emitting x_i in state l.

A mathematical derivation for the above equation is similar to that of $v_l(i)$ for Viterbi. Just replace the max with a $\sum$. The forward evaluation algorithm is shown below:



- Initialization
  $f_0(0) = 1$, $f_k(0) = 0$ for $k > 0$
- Main iteration
  for $i = 1...n$
    $f_l(i) = e_l(x_i).\Sigma_k\,(f_k(i-1).a_{kl})$
- Termination
  $p(x) = \Sigma_k f_k(n)a_{k0}$

Figure 1: Forward evaluation

Again, if there is no end state, then the term a_{k0} in computing $p(x)$ is omitted.

Backward evaluation

We revisit here the question of obtaining the most probable path. We know that we can compute the most probable path and its probability using the Viterbi algorithm. What we question here is whether the most probable path is actually what we want. What if there are many paths with almost the same high probability? Then the most probable path is not necessarily the one we want.

The most probable path might not be the most appropriate basis for further inference about the sequence $x_1...x_n$. For instance, we might want to know the most probable state for an observation x_i, i.e. what is the highest probable state that produced x_i? More generally, we are interested in finding $p(\pi_i = k|x_1...x_n)$. Let's see how to compute this:

$$p(\pi_i = k|x) = \frac{p(\pi_i = k, x)}{p(x)}$$

We know how to compute $p(x)$ using the forward algorithm. Let's see how we can compute $p(\pi_i = k, x)$.

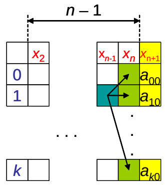
$$\begin{aligned} p(\pi_i = k, x) = & \; p(x_1...x_i, \pi_i = k)p(x_{i+1}...x_n|x_1...x_i, \pi_i = k) \\ = & \; p(x_1...x_i, \pi_i = k)p(x_{i+1}...x_n|\pi_i = k) \\ = & \; f_k(i).b_k(i) \end{aligned}$$

The first term is just $f_k(i)$ and denote the second term by $b_k(i)$. $b_k(i)$ is the probability of obtaining $x_{i+1}...x_n$ given that we are in state k at x_i. $b_k(i)$ can be computed in terms of $b_k(i+1)$ as shown below, hence the name Backward evaluation.

$$b_k(i) = \sum_l a_{kl} e_l(x_{i+1}) b_k(i+1)$$

This is saying that the probability of obtaining $x_{i+1}...x_n$ given that we start in state k is equal to the probability of transitioning from state k to some state l, emitting x_{i+1} in state l, and then obtaining $x_{i+2}...x_n$ given that we start in state l (note how we are computing it backward). The following figure illustrates the backward evaluation algorithm.

- Initialization
  $b_k(n) = a_{k0}$ for all k
- Main iteration
  for $i = n - 1 \ldots 1$
  $b_k(i) = \Sigma_l\, a_{kl} e_l(x_{i+1}) b_l(i+1)$
- Termination
  $p(x) = \Sigma_l\, a_{0l} e_l(x_1) b_l(1)$



Time = $O(k^2n)$
Space = $O(kn)$

Figure 2: Backward evaluation algorithm

Problem of very small numbers

The decoding and evaluation algorithms presented so far suffer from a severe practical problem which is that multiplying many probabilities always yields very small numbers that will give underflow error on any computer. We can solve this problem by performing the algorithms in log space (transforming multiplication into summation).

log space Viterbi
Let $V_l(i) = \log v_l(i)$. Then

$$\begin{aligned} V_l(i) = &\ \log(e_l(x_i)\max_k(v_k(i-1)a_{kl})) \\ = &\ \log e_l(x_i) + \log \max_k(v_k(i-1)a_{kl}) \\ = &\ \log e_l(x_i) + \max_k \log(v_k(i-1)a_{kl}) \\ = &\ \log e_l(x_i) + \max_k(\log v_k(i-1) + \log a_{kl}) \\ = &\ \log e_l(x_i) + \max_k(V_k(i-1) + \log a_{kl}) \end{aligned}$$

log space Forward
Let $F_l(i) = \log f_l(i)$. Then

$$\begin{aligned} F_l(i) = &\ \log(e_l(x_i)\textstyle\sum_k(f_k(i-1)a_{kl})) \\ = &\ \log e_l(x_i) + \log \textstyle\sum_k(f_k(i-1)a_{kl}) \\ \neq &\ \log e_l(x_i) + \textstyle\sum_k \log(f_k(i-1)a_{kl}) \\ but = &\ \log e_l(x_i) + \log \textstyle\sum_k 2^{F_k(i-1)+\log a_{kl}}\ (assuming\ \log\ base\ 2) \end{aligned}$$

HMM learning

Probably the most difficult problem faced when using HMMs is that of specifying the model in the first place. There are two parts to this of course: the design of the structure, what states there are and how they are connected; and the assignment of parameter values, the transition and emission probabilities a_{kl} and $e_k(b)$. In this section we discuss the parameter estimation problem.

Let θ be the parameters of the HMM (transition probabilities and emission probabilities, i.e. all the a's and e's). Our framework for estimation will be as follows: Given independent *training* sequences x^1, ..., x^n, we would like to find θ that will maximize $p(x^1...x^n|\theta) = \prod_i p(x^i|\theta)$. In this case, θ will be called the maximum likelihood parameters (it maximizes the likelihood of obtaining the training sequences). The rational is the following: if we believe that these sequences should be obtained by the model, let's find the parameters that will maximize the probability of obtaining the sequences.

State sequence is known

Assume that the state path for each x^i is known. For instance, we have sequences where all CpG islands are labeled. This is often the case with the estimation problem. Since the paths are known, we can compute A_{kl}, the number of transitions from state k to state l in the training sequences, and $E_k(b)$, the number of times b was emitted in state k in the training sequences. Then we can shown that the maximum likelihood parameters are given by:

$$a_{kl} = \frac{A_{kl}}{\sum_{l'} A_{kl'}}$$

$$e_k(b) = \frac{E_k(b)}{\sum_{b'} E_k(b')}$$

What we are basically doing here is estimating the probabilities from counts. We now prove that this estimation produces the maximum likelihood probabilities. Assume that we are looking at the outcomes of n independent observations and we count n_i observations for outcome i, $\sum_i n_i = n$. Consider the parameters $\theta^{ML} = \{\theta_i^{ML} = \frac{n_i}{n}\}$ assigning a probability $\frac{n_i}{n}$ for outcome i. We need to prove that $p(x_1...x_n|\theta^{ML}) > p(x_1...x_n|\theta)$ for any $\theta \neq \theta^{ML}$. It will be equivalent to prove that $\log \frac{p(x|\theta^{ML})}{p(x|\theta)} > 0$.

$$\begin{aligned}
\log \tfrac{p(x|\theta^{ML})}{p(x|\theta)} &= \log \frac{\prod_i (\theta_i^{ML})^{n_i}}{\prod_i \theta_i^{n_i}} \\
&= \textstyle\sum_i n_i \log \frac{\theta_i^{ML}}{\theta_i} \\
&= \textstyle\sum_i n\theta_i^{ML} \log \frac{\theta_i^{ML}}{\theta_i} \\
&= n \textstyle\sum_i \theta_i^{ML} \log \frac{\theta_i^{ML}}{\theta_i} > 0
\end{aligned}$$

The last summation is the relative entropy of θ^{ML} and θ which is always positive, and 0 iff $\theta^{ML} = \theta$ (information theory).

There are some problems associated with this approach: maximum likelihood estimation is vulnerable for overfitting if we have insufficient data. For example, if state k was never used in the set of training sequences, then a_{kl} will be zero for all l and similarly $e_k(b)$ will be zero for all b. In order to avoid such a situation, we can start with some pseudocounts r_{kl} for A_{kl} and $r_k(b)$ for $E_k(b)$. Large pseudocounts indicate a strong prior belief about the probabilities (it will require more data to modify it), while small pseudocounts are used to avoid zero probabilities. The new formulas for a_{kl} and $e_k(b)$ are given as follows:

$$a_{kl} = \frac{A_{kl} + r_{kl}}{\sum_{l'} A_{kl'} + r_{kl'}}$$

$$e_k(b) = \frac{E_k(b) + r_k(b)}{\sum_{b'} E_k(b') + r_k(b')}$$

Consider the example of the dishonest casino. We can start with the following pseudocounts:

$$\begin{array}{ll}
r_{0F} = r_{0L} = r_{F0} = r_{L0} = 1 & [\textit{avoid zero probability}] \\
r_{FL} = r_{LF} = r_{FF} = r_{LL} = 1 & [\textit{avoid zero probability}] \\
r_F(1) = r_F(2) = ... = r_F(6) = 20 & [\textit{strong belief that fair is fair}] \\
r_L(1) = r_L(2) = ... = r_L(6) = 5 & [\textit{wait and see how loaded it is}]
\end{array}$$

State sequence is unknown

What if a new species with different distribution of CpG islands comes in? Although we can obtain some genomic sequences for the new species, these genomic sequences are not labeled. How can we train the HMM to recognize the new CpG islands?

This is an example of training HMMs without knowing the actual state paths. In general, we need to find the maximum likelihood parameters θ for a set of training sequences without knowing the paths.

A famous training algorithm is known as the Baum-Welsh algorithm and is depicted below:

Baum-Welsh training

```
start at iteration 0 with some $\theta$, call it $\theta^0$
$L^0 \leftarrow \log p(x^1, ..., x^n|\theta^0)$
$i \leftarrow 0$
**repeat**
    $i \leftarrow i + 1$
    $A^i_{kl} \leftarrow E[A_{kl}|x^1, ..., x^n, \theta^{i-1}]$   (expected value)
    $E_k(b)^i \leftarrow E[E_k(b)|x^1, ..., x^n, \theta^{i-1}]$   (expected value)
    calculate $\theta^i$ using maximum likelihood estimators from counts $A^i_{kl}$ and $E_k(b)^i$ as before
    $L^i \leftarrow \log p(x^1, ..., x^n|\theta^i)$   (new likelihood)
**until** $L^i - L^{i-1} <$ *threshold*
```

The Baum-Welsh algorithm start with an arbitrary θ and iteratively computes a new θ^{i+1} based on estimation from counts A_{kl} and $E_k(b)$, which are computed as the expected value of the corresponding counts using θ^i. The algorithm stops when the difference in the likelihood of the system is below a fixed threshold. Baum-Welsh is a special case of a general algorithm known as Expectation Maximization (EM). EM guarantees that $p(X|\theta^{i+1}) \geq p(X|\theta^i)$. It will therefore converge to a local maximum (not necessarily the global maximum).

The heart of the Baum-Welsh algorithm is to compute $E[A_{kl}|x^1, ..., x^n, \theta^{i-1}]$ and $E_k(b)^i \leftarrow E[E_k(b)|x^1, ..., x^n, \theta^{i-1}]$. Let's see how these two expectations can be computed. We will start with the first one.

$$\begin{aligned}
E[A_{kl}|x^1, ..., x^n, \theta] &= \textstyle\sum_j E[A^j_{kl}|x^j, \theta] && \textit{by linearity of expectation} \\
E[A^j_{kl}|x^j, \theta] &= \textstyle\sum_i E[\#\ of\ k \rightarrow l\ at\ x^j_i|x^j, \theta] && \textit{by linearity of expectation} \\
E[\#\ of\ k \rightarrow l\ at\ x^j_i|x^j, \theta] &= p(k \rightarrow l\ at\ x^j_i|x^j, \theta) && \textit{indicator random variable} \\
p(k \rightarrow l\ at\ x^j_i|x^j, \theta) &= p(\pi_i = k, \pi_{i+1} = l|x^j, \theta) \\
p(\pi_i = k, \pi_{i+1} = l|x^j, \theta) &= p(\pi_i = k, \pi_{i+1} = l, x^j|\theta)/p(x^j|\theta) \\
p(\pi_i = k, \pi_{i+1} = l, x^j|\theta) &= f^j_k(i)a_{kl}e_l(x_{i+1})^j b^j_l(i+1)
\end{aligned}$$

Where A^j_{kl} is the expected number of transitions from k to l in sequence x^j, $f^j_k()$ and $b^j_k()$ are the forward and backward probabilities at state k for x^j.

Putting all of the above together, we obtain:

$$E[A_{kl}|x^1, ..., x^n, \theta] = \sum_j \frac{1}{p(x^j|\theta)} \sum_i f^j_k(i)a_{kl}e_l(x^j_{i+1})b^j_l(i+1)$$

Similarly, we can show that

$$E[E_k(b)|x^1, ..., x^n, \theta] = \sum_j \frac{1}{p(x^j|\theta)} \sum_{i|x^j_i = b} f^j_k(i)b^j_k(i)$$

Another training algorithm that avoid employing the forward and backward algorithms is known as Viterbi training, depicted below:

Viterbi training

```
start at iteration 0 with some $\theta$, call it $\theta^0$
$i \leftarrow 0$
**repeat**
    $i \leftarrow i + 1$
    $A^i_{kl} \leftarrow$ number of transitions $k \rightarrow l$ on the most probable paths for $x^1, ..., x^n$ using $\theta^{i-1}$
    $E^i_k(b) \leftarrow$ number of times $k$ emits $b$ on the most probable paths for $x^1, ..., x^n$ using $\theta^{i-1}$
    calculate $\theta^i$ using maximum likelihood estimators from counts $A^i_{kl}$ and $E_k(b)^i$ as before
**until** none of the optimal paths changes
```

Although Viterbi training avoids performing the forward and backward algorithms, it performs less well than Baum-Welsh in general. The reason for this being that Viterbi training only considers the counts on the optimal paths. In fact, it is not maximizing the true likelihood $p(x^1, ..., x^n|\theta)$ anymore. It makes sense to use Viterbi training for simplicity, especially if one is only using Viterbi decoding to infer knowledge about the sequences.

References

Durbin R. et al, Biological Sequence Analysis, Chapters 3 & 11.