Introduction to Computational Biology
Lecture # 3: Estimating Scoring Rules for Sequence Alignment

Yehonatan Sela
13/11/2007

1 Brief Review

In the previous lecture we continued our discussion of sequence alignment algorithms. Specifically we used the divide and conquer method to come up with a space efficient approach to sequence alignment. We went on to explain how our algorithm can be modified to enable local alignments, finishing with heuristic methods for improving time and space efficiency and its use for BLAST search.

2 A Probabilistic Model

In our discussion of sequence alignment algorithms we assumed we have a given set of scoring rules for generating the alignment. Today we will discuss a way for generating these scoring rules.

2.1 Two different approaches

It is possible to create a scoring matrix by a calculated selection of criteria according to any arbitrary set of biological constraints. Yet, we must realize that there are countless constraints to keep in mind and once we have generated this matrix according to a chosen set of criteria we hardly have any assurances as to its success in estimating the alignment score.

We would rather create the matrix in accordance with some methodology that will give some indication to its success in estimating the likelihood of an alignment. For this we use a training set of "real" alignments.

Two models exist:

1. Generative method - Modelize the way in which the generated data of the frequencies in the training set is translated to a score.

2. Discriminative method - Choose a score that prefers the training alignments from alternatives.

Today, as well as along the course, we will focus our discussion on the first approach.

2.2 Building the model

Lets assume we can divide our analysis of the problem into two disjoint complimentary occurrences:

1. M - The sequences are evolutionarly related $P(\bar{x}, \bar{y}|M)$

2. R - The sequences are unrelated $P(\bar{x}, \bar{y}|R)$

Lets consider the latter case first.

We will assume that the value of a given position in the sequence is independent of adjacent positions in the sequence. In addition, when the sequences are unrelated (R), we can assume that $\bar{x}, \bar{y}$ at any position $i$ are independent of each
other.
In other words, for any position \( i \), both \( x_i \) and \( y_i \) are sampled independently from some background distribution \( q(\cdot) \).
So that the likelihood of the given \( \vec{x}, \vec{y} \) to be unrelated is:

\[
P(\vec{x}, \vec{y} | R) = \prod_{i=1}^{n} q(x_i) \prod_{i=1}^{n} q(y_i)
\]  

(1)

In the first case above, we assume the two sequences are related, so they evolve from a common ancestor. For simplicity we will continue assuming that each position \( i \) in \( (x_i, y_i) \) is independent of the others. So if we assume \( x_i, y_i \) are sampled from some distribution \( p(\cdot, \cdot) \) of letter pairs. The probability that any two letters \( a, b \) evolved from some ancestral letter is \( p(a, b) \). So the likelihood of the given \( \vec{x} \vec{y} \), to be related is:

\[
P(\vec{x}, \vec{y} | M) = \prod_{i=1}^{n} p(x_i, y_i)
\]

(2)

### 2.3 A decision problem

So now we have stumbled across a decision problem. Given the two sequences \( \vec{x}, \vec{y} \) we have to decide whether they are sampled from \( M \) or from \( R \). We want to construct a decision procedure \( D(\vec{x}, \vec{y}) \) that returns \( M \) or \( R \). Basically we want to compare the likelihood of our data in both models.

First, lets notice that our decision procedure can make either one of two error types:

- **Type I** - \( \vec{x}, \vec{y} \) are sampled from \( R \) but \( D(\vec{x}, \vec{y}) = M \)
- **Type II** - \( \vec{x}, \vec{y} \) are sampled from \( M \) but \( D(\vec{x}, \vec{y}) = R \)

The probabilities of such errors are also defined:

- \( \alpha(D) = P(D(\vec{x}, \vec{y}) = M | R) \)
- \( \beta(D) = P(D(\vec{x}, \vec{y}) = R | M) \)

We would of course favor a procedure which minimizes both error types.

**Definition 2.1** Given two decision rules \( D_1, D_2 \) we would say that \( D_2 \) is dominated by \( D_1 \) if \( \alpha(D_1) < \alpha(D_2) \) and \( \beta(D_1) < \beta(D_2) \).

Lets examine decision rules of the form:

\[
D_k(\vec{x}, \vec{y}) = \begin{cases} 
M & \frac{P(\vec{x}, \vec{y} | M)}{P(\vec{x}, \vec{y} | R)} \geq k \\
R & \text{else}
\end{cases}
\]

The following statistics lemma gives us an interesting result:

**Lemma 2.2** (Neyman-Pearson) - For any \( k \) and any \( D_k \) of the above form and let \( D' \) be another decision rule, then if \( \alpha(D') \leq \alpha(D_k) \) then it follows that \( \beta(D') \geq \beta(D_k) \).

Basically the lemma tells us that no decision rule dominates a rule of the type \( D_k \) defined above. No rule is actually "much" better then the above rule. Thus we can restrict our discussion to rules of the above form.

From plugging previous equations in the model we get:

\[
\frac{P(\vec{x}, \vec{y} | M)}{P(\vec{x}, \vec{y} | R)} = \prod_i \frac{p(x_i, y_i)}{q(x_i)q(y_i)} = \prod_i \frac{p(x_i)q(y_i)}{q(x_i)q(y_i)}
\]

Or for convenience, by taking a logarithm from both equation sides, of the form:
\[
\log \frac{P(\vec{x}, \vec{y}|M)}{P(\vec{x}, \vec{y}|R)} = \log \frac{\prod_i p(x_i, y_i)}{\prod_i q(x_i)q(y_i)} = \sum_i \log \frac{p(x_i, y_i)}{q(x_i)q(y_i)}
\]

Now we can define our scoring rule matrix as follows:

\[
\sigma(a, b) = \log \frac{p(a, b)}{q(a)q(b)}
\]

### 2.4 Estimating Probabilities

If we could estimate the probabilities \(p(\cdot, \cdot)\) and \(q(\cdot)\) from our data we would have our scoring matrix \(\sigma(\cdot, \cdot)\).

We will discuss parameter estimation through an example:

**Example 2.3 Thumbtack Throwing**

When tossing a thumbtack, it can land on one of two positions, Head or Tail with an unknown probability.

**The estimation task:**

*Given a sequence of toss samples \(x_1, x_2, \ldots, x_m\) we would like to estimate the probabilities \(P(H) = \theta\) and \(P(T) = 1 - \theta\).*

Assuming the thumbtacks are identical and that they are thrown in the same way each time, it is pretty safe to say that samples are independent and identically distributed (i.i.d.). That’s why given \(P(H) = \theta\) leads to \(P(x_1, \ldots, x_N : \theta) = \prod_i^N P(x_i)\)

We can obviously see that the tack throwing is a binomial experiment so it is distributed binomially: \(P(#H = k) = \binom{N}{k} \theta^k (1 - \theta)^{N-k}\)

Thus the expectation is \(E[\#H] = \frac{\#H}{N}\) and we intuitively realize that we can estimate \(\theta\) by \(\frac{\#H}{N}\).

We would like to generalize a process of finding an estimator. one method is the MLE - Maximum Likelihood Estimator.

Basically what we need for finding an estimator \(\hat{\theta}\) that will maximize \(L(\theta) \Rightarrow \hat{\theta} = \arg \max_\theta L(\theta)\) is:

1. Build a likelihood function \(L\).
2. Find a maximum for the likelihood function.

For example in the thumbtack problem we could define the likelihood function:

\[
L(\theta|D) = \theta^{N_H} (1 - \theta)^{N_T}
\]

Where \(N_H, N_T\) are defined \(N_H = \sum_i 1_{x_i = H}\), \(N_T = \sum_i 1_{x_i = T}\)

We would also like to point out that \(N_H, N_T\) are sufficient statistics.

**Definition 2.4** \(S(D)\) is a sufficient statistic if for any two datasets \(D, D'\): \(S(D) = S(D') \Rightarrow L(\theta|D) = L(\theta|D')\).

This means that the transformation \(S\) keeps all the minimal necessary information to compute the likelihood.

Now we need to find the maximum of our likelihood function, by finding \(L'(\theta) = 0\) and \(L''(\theta) < 0\).

It’s often useful looking at the likelihood log function which we denote by \(\ell(\theta)\).

For example in the above thumbtack problem:

\[
\ell(\theta) = \log L(\theta) = N_H \log(\theta) + N_T \log(1 - \theta)
\]

\[
\ell' = \frac{N_H}{\theta} - \frac{N_T}{1 - \theta} = 0 \Rightarrow \theta = \frac{N_H}{N_H + N_T}
\]
2.5 Langrange Multipliers (aka Coefficients)

When we have a problem of maximization of \( f(x) \) but we have some constraint \( C(x) = 0 \), we will define a new Lagrangian function

\[
J(x, \lambda) = f(x) - \lambda c(x)
\]

and we will want to show that when the partial derivative of \( x \) and \( \lambda \) is zero - the constraint is satisfied and we are in stationary point.

\[
\frac{\partial J}{\partial \lambda} = C(x) = 0
\]

our constraint is satisfied and

\[
\frac{\partial J}{\partial x} = \frac{\partial f(x)}{\partial x} - \lambda \frac{\partial c(x)}{\partial x} = 0 \Rightarrow \frac{\partial f(x)}{\partial x} = \lambda \frac{\partial c(x)}{\partial x}
\]

and in case we have more then one constraint

\[
J = f(x) - \sum_j \lambda_j \frac{\partial C_j(x)}{\partial x}
\]

and in the maximum we get

\[
\frac{\partial f(x)}{\partial x} = \sum_j \lambda_j \frac{\partial C_j(x)}{\partial x}
\]

We showed that the derivation according to \( x \) of \( f(x) \) is a linear combination of the gradients of the constraints - we are in a point on the line of the constraint, where you can go no further, in the direction of \( f(x) \) (if we are going on the line of the constraint as long as we go transversally to the contour line of \( f(x) \) we are going ‘uphill’, but when we touch it tangentially we can go no further - ‘top of the hill’. We know we got to this point when the two derivatives are the same).

![Figure 1: Drawn in green is the locus of points satisfying the constraint \( g(x,y) = c \). Drawn in blue are contours of \( f \). Arrows represent the gradient, which points in a direction normal to the contour.](http://en.wikipedia.org/wiki/Lagrange_multipliers)

Now we are ready to go back to our problem.
2.6 Back to our problem

We still need to estimate the letter probabilities in our sequence.

2.6.1 Finding \( q(\cdot) \)

In principle, that’s exactly like the thumbtack problem, besides the fact the data is Multinomial and not Binomial, so we have a constraint.

Now we have 1, \ldots, \( k \) possibly different probabilities for each single letter we shall mark them \( \vec{\theta} = (\theta_1, \ldots, \theta_k) \).

So using our previous assumptions about independence between the positions and the sequences, lead our likelihood function to be:

\[
L(\vec{\theta}) = N \prod_{j=1}^{N} p(x_j) = \prod_{k} \theta_k^{M_k} \Leftrightarrow
\]

Where \( M_k = \sum_j 1_{x_j = k} \) and \( \forall k (0 \leq \theta_k \leq 1) \) and \( \sum_k \theta_k = 1 \)

\[
\Leftrightarrow \ell(\vec{\theta}) = \sum_k N_k \log(\theta_k)
\]

Since we want to find local maximums of the averages, we will use Lagrange Multipliers. For the general case (as shown before):

\[
\arg \max_{\vec{\theta}} f(\vec{\theta}), s.t.c(\vec{\theta}) = 0
\]

\[
J(\vec{\theta}, \lambda) = f(\vec{\theta}) - \lambda c(\vec{\theta}), \forall i \frac{\partial J}{\partial \theta_i} = 0 \Leftrightarrow \frac{\partial f}{\partial \theta_i} = \lambda \frac{\partial c}{\partial \theta_i}
\]

\[
\frac{\partial J}{\partial \lambda} = 0 \Leftrightarrow c(\vec{\theta}) = 0
\]

Now lets solve our problem again:

\[
J = \sum_k N_k \log(\hat{\theta}_k) - \lambda(\sum_k N_k - 1)
\]

\[
\frac{\partial J}{\partial \hat{\theta}_k} = \frac{N_k}{\hat{\theta}_k} - 1 \Rightarrow N_k - 1 = 0 \Rightarrow \hat{\theta}_k = \frac{N_k}{\lambda}
\]

\[
0 = \sum_k \hat{\theta}_k - 1 = \sum_k \frac{N_k}{\lambda} - 1 \Rightarrow \lambda = \sum_k N_k \Rightarrow \hat{\theta}_k = \frac{N_k}{\sum_k N_k} = \frac{N_k}{N}
\]

2.6.2 Finding \( p(\cdot, \cdot) \)

We will examine two methods of estimating \( p(\cdot, \cdot) \), PAM and BLOSUM:

**PAM - Percent Accepted Mutations**  Very similar sequences can easily be aligned using naive means, and the differences between these sequences may be studied.

We shall define 99% of identity as one unit of pam (u.o.p), or one unit of time (u.o.t). That is 1 mutation every 100 nucleotides. It should be noted that 1 u.o.p does not correspond to any given time period.

The information about the individual kinds of mutations and about the relative mutability of the amino acids or nucleotides can be combined into one distance-dependent “mutation probability matrix”. The elements of this matrix give the probability that the amino acid or nucleotide in one column will be replaced by the amino acid or nucleotide in some row after a given evolutionary interval we defined a 1 u.o.p.
This process (for each position independently) is defined as \textit{Markov Chain}. A chain is defined by the transition probability \( P(X^{t+\Delta} = b | X^t = a) \) - the probability that the next state is \( b \) given that the current state is \( a \).

These probabilities are often described by a matrix:

\[
T[\Delta]_{ab} = P(X^{t+\Delta} = b | X^t = a)
\]

Based on the \( T[X] \) matrix, we can compute the probabilities of changes over 2 u.o.p:

\[
P(X^{t+2\Delta} = b | X^t = a) = \text{(complete probability equation)} = \sum_c P(X^{t+\Delta} = b, X^{t+\Delta} = c | X^t = a) = \text{(Conditional probability equation)} = \sum_c P(X^{t+\Delta} = b | X^{t+\Delta} = c, X^t = a) \sum_c P(X^{t+\Delta} = c | X^t = a) = \sum_c T_{ac} \cdot T_{cb}
\]

Figure 2: The left matrix is of size \( a \times c \), and the right one is of size \( c \times b \) so the product matrix is of size \( a \times b \).

And this is actually matrices multiplication thus \( T[2\Delta] = T[\Delta] \cdot T[\Delta] \) and by induction we get:

\[
T[k\Delta] = T[\Delta]^k
\]

For example, a matrix with an evolutionary distance of 0 PAMs would have ones on the main diagonal and zeros elsewhere. A matrix with an evolutionary distance of 1 PAM would have numbers close to one on the main diagonal and small numbers off the main diagonal.

\textbf{PAM1}: \( P_1(x_i, y_i) \) = The probability \( x, y \) diverged from a single ancestor 1 u.o.p apart.

In order to calculate the next PAM matrix:

\[
P_2(x, y) = \sum_y P_1(x, y, z) = \sum_y P_1(x, y) P_1(z | y) = p(x) \sum_y P_1(y | x) P_1(z | y)
\]

It is common to use PAM-250 matrices. At this evolutionary distance (250 substitutions per hundred residues) only one amino acid in five remains unchanged and the percent divergence has increased to roughly 80.

However, PAM has several problems as it assumes a \textit{Markov Chain} on long time substitutions where in reality such processes happen only on short time ones. As a replacement might occur in both direction and the replacement back won’t be considered, and two repetitive changes might occur in a single amino acid, and this will be considered only as one change. Not to mention the fact it doesn’t do anything with gap opening.

To overcome these bias came the BLOSUM method:
**BLOSUM - BLOck SUBstitution Matrix**  
This method is newer than PAM and it is based on the idea that a certain percent of conservation between two sequences defines the evolutionary distance between them. For a certain value of evolutionary distance, the calculation of $p_N(a, b)$ is based on the same principle as the ones discussed above, with the exception that this time, we only count events that might occur during a defined evolutionary time range.

All sequences related to each other by that range are clustered together and dealt like a single sequence. Then, we sum over these cluster representatives and for each block pair - if the similarity between them exceeds some rate they contribute another ‘one’ to the sum.

For example, a BLOSUM62 matrix is calculated from protein blocks such that if two sequences are more than 62% identical, then the contribution of these sequences is weighted to sum to one.