Algorithms in Computational Biology
Lecture # 6: MLE, Pairwise Alignment Score Functions,Heuristics,FASTA,BLAST

Dolev Rahat

November 28, 2016

1 Maximum Likelihood Estimation

Suppose we have a series of N coin flips $D \in \{H,T\}^N$. For an individual flip we have $x_i \sim Ber(\theta)$ with $x_i = \begin{cases} 1 & ith\, tossing\, yields\, head \\ 0 & otherwise \end{cases}$. To estimate θ we will maximize the likelihood $L(\theta;D) = P(D|\theta) = \theta^{N_H} \cdot (1-\theta)^{N_T}$, where N_H is the number of heads and N_T the number of tails. In the previous lecture we saw that by differentiating the likelihood and equating the derivative to zero we get $\theta_{MLE} = \frac{N_H}{N_H+N_T}$.

We will now extend the maximum likelihood estimation to the multinomial case. Suppose that we now throw a die N times. We will again get a series of outcomes $D \in \{1,2,3,4,5,6\}^N$. We will mark with $N_1,...,N_6$ the number of throws in which we got each digit.

We now have $D \sim Mult(\theta)$ with $\theta = \begin{bmatrix} \theta_1 \\ ... \\ \theta_6 \end{bmatrix}$, where θ_i is the probability to get i in a given throw.

The likelihood is:

$$L(\theta;D) = \Pi_{i=1}^{6} \theta_i^{N_i} \tag{1}$$

It would be more convenient to us to maximize the log-likelihood:

$$LLR(\theta;D) = \Sigma_{i=1}^{6} N_i \cdot log(\theta_i) \tag{2}$$

Since the log-likelihood function is monotonically increasing, the value of θ maximizing the log-likelihood is also the maximum of the likelihood itself. We would like to maximize the log likelihood by differentiating with respect to θ as we did in the binomial case. However, the multinomial case is a little more complicated because we now need to satisfy two additional constraints:

$$\Sigma_{i=1}^{6}\theta_i = 1 \tag{3}$$

$$\forall i \in [6] : \theta_i \geq 0 \tag{4}$$

Note that since θ_i is a probability, we actually need to satisfy $\forall i : \theta_i \in [0, 1]$, but given the first condition it is enough to require that the parameters are non-negative in order to satisfy this. In fact since we are maximizing the log likelihood we will not need to explicitly formulate even this form of constraint (4) as the log function is not defined over negative values.

In order to maximize the likelihood under these constraints we will use a Lagrange multiplier. Lagrange multipliers follow the general form:

$$J(\vec{x}, \vec{\lambda}) = f(x) - \Sigma_j \lambda_j C_j(x) \tag{5}$$

Where f is the function being maximized, λ a vector of weights and C a vector of constraints.

We can transform constraint (3) to the functional form:

$$C(\vec{\theta}) = \Sigma_{k=1}^{6}\theta_k - 1 \tag{6}$$

By substituting equations (1) and (6) into equation (5) we get:

$$J(\vec{\theta}, \lambda) = \Sigma_{i=1}^{6} N_i \cdot log(\theta_i) - \lambda(\Sigma_{k=1}^{6}\theta_k - 1) \tag{7}$$

Note that the monotonicity of the log likelihood function also means that we do not need to explicitly account for constraint (4), as negative values of θ_i will always achieve lower values of L then non-negative values.

By differentiating $J(\vec{\theta}, \lambda)$ with respect to both θ and λ (see previous lecture for details) we get:

$$\theta_i^{MLE} = \frac{N_i}{N}$$

2 Pairwise Alignment Score Functions

Suppose we are given two sequences a, b and want to decide between the two following hypotheses:

H_0 - the sequences are independent.

H_1 - the sequences have a common ancestor.

From Neyman-Pearson lemma we know that the optimal test for deciding between the hypotheses is the likelihood ratio test:

$$\sigma(a, b) = log\frac{P_1(a, b)}{P_0(a) \cdot P_0(b)}$$

In practice we can use pairwise alignment with the following scoring function: $\sigma(a,b) = \Sigma_{i,j} log \frac{P_1(a_i,b_j)}{P_0(a_i)\cdot P_0(b_j)}$. Where $P_1(a_i,b_j)$ is the probability to find position a_i aligned to position b_j under the assumption that the sequences indeed have a common ancestor. To estimate the independent probabilities $P_0(a_i), P_0(b_j)$ we can use a database of sequences and find the frequencies of the residues a_i and b_j.

Caveats:

1. We need to make sure that we use a database that is relevant to the sequences under study. If we use a database with sequences that did not originate from a **biological context** that is relevant to our sequences it is also likely that the distribution of residues in the database will be different from the distribution from which the sequences that we study were drawn, and then the estimates that we will obtain may be misleading.

2. The database may contain redundant sequences, for example paralogous[1] sequences that originated from a gene duplication event. Under such conditions, the occurrence of the residues in the database will not be i.i.d., and the estimates we will obtain from the database will not be consistent with the **assumption of independence**.

3 Heuristic algorithms: FASTA and BLAST

Motivation: Suppose we are given a protein sequence Q of $n = 10^3$ amino acids (AAs), and a database DB with approximately 10^6 protein sequences with a mean length of $n = 10^3$ AAs, and are asked to align Q against all sequences $P \in DB$. Recall that using dynamic programming, the time complexity of this computation is $O(n^2|P|)$, or something in the order of $10^6 \cdot 10^3 \cdot 10^3$ computations. Even using a modern computer this will take a few hours. Therefore, we want a faster way to find the sequences in DB that are similar to Q. To this end we will use heuristic algorithms, which are algorithms that do not guarantee an optimal solution. In this case we will require a preliminary match of several AAs/bases between $P \in DB$ and Q as a prerequisite for pursuing further alignment of the two sequences. In other words, we are willing to sacrifice some of the sensitivity of dynamic programming in favor of speed.

[1] Paralogs - genes/proteins with a common ancestor that are found **in the same species.** As opposed to orthologs which are genes/proteins with a common ancestor in different species.

3.1 FASTA (Pearson and Lipman 1985)

Assumptions:

1.Q and P share a common identical sub-sequence.

2. This sub-sequence can be extracted while permitting only small gaps in the alignment between Q and P.

In other words, if we plot the dot matrix of Q and P we will see stretches of identical positions between the sequences, as depicted in Fig. 1B.

Input:

query sequence Q; reference database DB; integer *ktup* indicating length of seed (preliminary match). Alternatively, the input may be a pre-defined list of seed sequences of interest.

Algorithm

0. Preprocessing: For every $P \in DB$ we construct a hash table indicating for each seed of length *ktup* its start coordinates in P. If there are several occurrences of s in P all of them will be reported. If the input is a list of seeds, the hash table will report the start coordinates of all seeds that are represented in P.

Note that this step should only be performed once for the entire database and need not be repeated every time the database is queried with a new sequence.

1. Build a similar hash table for Q.

2. Iterate over the hash tables of Q and P, for all seeds $s \in P \cap Q$ define i: start coordinate of s in Q, j: start coordinate of s in P, $offset(s) = j - i$ (Fig. 1A)

3. Compute distribution of offsets. Choose diagonal corresponding to the most frequent offset value (Fig. 1B).

4. Extend the seeds chosen in step (2) in both directions using matches and mismatches (no gaps) to find maximal ungapped matches (Fig. 1C).

5. Join ungapped matched regions using dynamic programming within a (narrow) bandwidth b around the diagonal[2] (Fig. 1D).

Time Complexity Analysis

Define $m := |DB|$, $b := bandwidth\ for\ extension\ of\ ungpped\ matches\ with\ gaps$

0. Preprocessing: Construction of hash tables for every $P \in DB$: $O(n \cdot m)$

1. Construction of the hash table for Q: $O(n)$

2. Calculating offsets: $O(n \cdot m)$

3. Finding most frequent diagonal: $O(n \cdot m)$

4. Extending matches: $O(n \cdot m)$

5. Adding gaps: $O(b^2 \cdot m)$ (but b is small)

Overall: $O(n \cdot m)$. In conclusion we reduced the runtime from $O(n^2 m)$ to $O(nm)$, with some expense in sensitivity.

[2]Note that the restriction to a narrow band is justified by the assumption that there exist an alignment of P and Q with only small gaps.

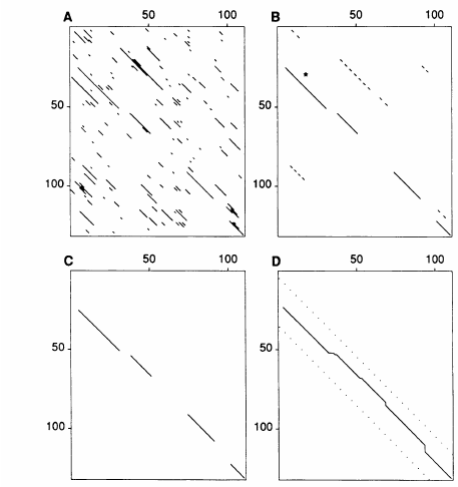

Figure 1: FASTA algorithm: (A) dot matrix for the matched seeds of P and Q. Each dot at cell (i,j) denotes a seed appearing in index i in P and in j in Q. (B) Selecting for frequent offsets. (C) Extending the seeds without gaps (D) Introducing gaps with dynamic programming Source: [1] fig. 1.

3.2 BLAST: Basic Local Alignment Search Tool (Altschul et al. 1990)

BLAST operates on principles similar to FASTA. The major difference is that it allows mismatches in the seed sequences. For example if Q contains the subsequence ACG, the sequences ACC and ACT are also potential seeds and will be represented in its hash table.

To generate the hash table we will decide on a similarity threshold T. next we will define for every $s \in P \, s.t. \, length(s) = ktup$

$$N_T(s) = \{v \in \Sigma^{ktup} \, s.t. \, \Sigma_i \sigma(v_i, w_i) \geq T\}$$

where Σ^{ktup} is the group of all strings of length $ktup$ from the alphabet of interest (such as nucleotides or amino acids). All sequences in $N_T(s)$ represented in P are added to the hash table. The hash table for Q is constructed in the same way. Note that allowing for mismatches in the seeds provides increased sensitivity compared to FASTA, with some increase in runtime. The level of sensitivity can be adjusted by the choice of T: lower T values will provide higher sensitivity but will also increase runtime, whereas increasing T will provide quicker performance but lower sensitivity.

Algorithm
Input: $Q, DB, ktup, T$
Preprocessing:
0.1 For every $P \in DB$ map all seeds to a hash table $H(P)$
0.2 For each seed $s \in \Sigma^{ktup}$ compute $N_T(s)$
1. For each seed s search the query sequence Q for seeds matching $N_T(s)$
Steps 2-5 identical to FASTA.

References

[1] William R Pearson and David J Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences*, 85(8):2444–2448, 1988.