Introduction to Computational Biology
Lecture # 12: The EM method
Matan Kubovský
19/2/2008

1 Introduction

In the previous lessons we introduced a probabilistic model which helped us to describe several types of biological sequences - The Hidden Markov Model. We also presented a problem in which we try to find a binding motif within a protein or DNA regulatory elements according to a complete data set. Building a HMM that displays a one-time appearance of the motif resulted in the OOPS (One Occurrence Per Sequence) model (see figure 1).

In this chapter we define a more perplexed problem, one which does not offer a complete data set, but a deficient one.

2 A short reminder

The OOPS model described in details in the previous lesson can be portrayed as follows:

![Figure 1: OOPS - One Occurrence Per Sequence.](image)

Having $\eta = \theta$ gives a uniform distribution for $p(l)$ and resulting in equal probability for both transitions $B1 \rightarrow 1$ and $B2 \rightarrow \text{End}$. We also defined $l$ to be the index of the last hidden state in the background, before the automaton entered the motif and finished our lesson with:

$$p(L = l|x_1, \ldots, x_N) \propto \prod_{j=1}^{k} \frac{p_j(x_{l+j})}{p_0(x_{l+j})}$$

where $p_0$ is the background distribution.
Now, in order to comprehend the likelihood of a certain sequence, we need to start with evaluating:

\[ p(x_1, \ldots, x_n) = \sum_l p(L = l | x_1, \ldots, x_n) \]  
(The Law Of Total Probability)

We would like to obtain the sufficient statistics in order to learn all \( p_j \)'s. This is feasible since we know the value of \( l \) for each sequence:

\[ N_{ja} = \sum_{m=1}^{M} 1 \cdot (x_{l \cdot j}^m[a] = a) \]

And the MLE is:

\[ \hat{p}_j(a) = \frac{N_{ja}}{M} = \frac{N_{ja}}{\sum_b N_{jb}} \]

This was quite simple since we know where the locations of \( l \) and \( j \). The problem is what happens if we don’t?

3 Evaluation of model parameters from an incomplete data set

3.1 The EM algorithm

The Expectation Maximization (EM) algorithm is a general algorithm for ML estimation with incomplete data. For HMM’s, the missing data are the hidden states, since we only know the observations and not the sequence of states producing them.

We would then calculate the probabilities \( p_1, \ldots, p_k \) on our own. In other words, we need a method for estimating \( N_{ja} \) and \( N_{jt} \). We’ll do so by calculating their expectations:

\[
E[N_{ja} | D] = \sum_m E[1 \cdot (x_{l \cdot j}^m[a] = a) | D] = \sum_m \sum_{l=1}^{n_m-k-1} p(L[m] = l | D) \cdot 1 (x_{l \cdot j}^m[a] = a) = \sum_m \sum_{l=1}^{n_m-k-1} p(L[m] = l | x_{1_{m}[m]}, \ldots, x_{n_{m}[m]}) \cdot 1 (x_{l \cdot j}^m[a] = a)
\]

The first equality is derived by the fact that the sum of the expectations is the expectation of the sum.

We would like to translate this evaluation into a modular algorithm:

3.2 MEME

MEME (Multiple EM for Motif Elicitation) is one of the most widely used tools for searching of peculiar ‘signals’ in sets of biological sequences. It includes the discovery of new transcription factor binding sites and protein domains and works by searching for repeated, ungapped sequence patterns that occur in the DNA or protein provided as input.

First, we will describe the guts of the algorithm - the MEME-step:
This section receives a profile and the data set as input \((p^0_j, D)\) and outputs a better profile according to D:

**MEME-step**\((p^0_j, D)\):

1. Initialization:
   1.1 \(N[j, a] = 0, \forall j = 1 \ldots k, \ a \in \{A, C, T, G\}\)
   1.2 \(l = 0\)

2. E-step:
   2.1 For \(m = 1 \ldots M\)
   2.1.1 \(p(l) = \prod_{j=1}^{k} \frac{p^0_{ij} [x_{i+j}[m]]}{p_0 [x_{i+j}[m]]} \quad \forall l \ldots n_m - k - 1\)
   2.1.2 \(Z = \sum_l p[l]\)
   2.1.3 \(l^+ = \log Z\)
   2.1.4 For \(l = 1 \ldots n_m - k - 1\)
   2.1.4.1 For \(j = 1 \ldots k\)
   2.1.4.1.1 \(N[j, x_{l+j}[m]] + = \frac{1}{Z} \cdot p[l]\)

3. M-step:
   3.1 \(p_j[a] = \frac{N[j, a]}{\sum_{a'} N[j, a']}\)
   3.2 Return \\{(p_j[j = 1 \ldots k], l)\}\n
The returned value is the new profile.

In order to unify these 3 steps into one method, we demonstrate a short encapsulated algorithm:

**MEME-loop** \\{(p_j[j = 1 \ldots k], D)\}

1 For \(t = 1 \ldots \infty\) (until convergence)
   1.1 \((\vec{p}^{(t)}, l^{(t-1)}) = MEME - step (\vec{p}^{(t-1)}, D)\)
   1.2 IF \((l^{(t-1)} - l^{(t-2)}) < \varepsilon\) then BREAK
   2 return \(\vec{p}^t\)

### 4 Discussion

After the detailed definition of the algorithm, there are several bothering questions we must confront:

1. What is the complexity and runtime of the algorithm?

Well, evaluation of the runtime of the MEME-step results in the value of:

\[
|D| \cdot k \leq M \cdot n \cdot k
\]

Since the size of our data is \(M \cdot n\)

Where,
- \(M\) - The number of sequences we have.
- \(n\) - The length of each sequence.
- \(k\) - The size of a motif.

Thus, resulting in runtime which is linear in the length of the input.
2. When will the algorithm stop running? (The Convergence issue):

We know that \( l(\vec{p}^t) \rightarrow l^* \).

This means that \( l \) is a monotonic function which converge into a certain limit. Seemingly, the likelihood converge, but this does not necessarily mean a convergence of the parameters. In order to cope with that, we must evaluate the likelihood in every MEME-loop iteration, check whether it changes within a value of \( \varepsilon \) and only then stop. Thus, in every iteration of the algorithm the likelihood will be changed in the following manner:

\[ l^* - l(\vec{p}^{t+1}) \approx C \cdot (l^* - l(\vec{p}^t)) \text{ where } C < 1 \]

Resulting in a typical saturation curve:

![Saturation Curve](image)

Figure 2: A typical saturation curve

Remarks:

(a) Tip of the day - Whenever you're using this kind of method, make sure that your data looks the same.

(b) It is possible to program a more sophisticated stopping condition which uses the above equation.

3. How do we initialize the parameters?

Given the fact that we have no prior information regarding the data, we must decide on a way to initialize \( p_0^t \):

There are several ways in order to do so, some are better:

(a) Use the uniform - background distribution: This solution does not suffice since we get a distribution which is quite close to the background distribution plus minor deviations resulting from the edges of the sequence.

(b) Take a random distribution: This means that we randomly pick a distribution from the space of distributions. The results from this solution can be either satisfactory or poor, depending on luck only.

(c) Consider all starting points: We can take as many different starting points as we want, run the algorithm and make a comparison between their likelihood. We will consider a high likelihood value as a better initialization point. In other words, we would uniformly choose a random distribution from our distribution space, attain the corresponding starting point and go on running the algorithm with the one which has the highest likelihood value.
To summarize this, let us consider all subsequences words of length $k$ from our text (there is a linear number of such these words - $M \cdot n$) and try them all. This method does not necessarily exhaust all possible profiles but it is worth a try.

The MEME algorithm mentioned before, takes advantage of the previous notion by considering all subsequences words in length $k$ and running the EM as described before initiated according to each one of them. Doing so, it returns the maximal value of likelihood found. This simple exhaustive method has a price - Quadratic run time.

4.1 Creating the dictionary

Before we end our little journey of EM we must confront another concerning question - Which word should we choose to be our motif?

Up until now we discussed and traveled through the space of profiles. To face with our next obstacle, we can start dealing with a discrete space - The space of motifs.

(a) A first solution for that would be taking a word randomly. If the motif exists in the majority of the sequences, there’s quite a good chance we will find it.

(b) Another solution could be creating a dictionary of more frequent words extracted from our sequences. This can be easily done by passing over all $k$-length words (There is only $M \cdot n$ of such these words) and increasing a counter whenever stumbling upon one. (i.e. using a hash table).

Advantages:
- Efficiency - In order to find in how many sequences a word appears we only need a linear time + the time to find the word in the dictionary. This means an efficient linear runtime (depands on the data structure chosen in order to represent the dictionary) which yields a fast procedure.
- Credibility - The chosen motif is quite promising since the words taken in mind were the most frequent in our data.

Disadvantages:
- Consensus - There is a possibility that for any sequence we might get the consensus word. Thus, missing the goal of finding a motif.
- Imprecision - Assume the motif appears within the sequences but with slight differences. Building a degenerate dictionary as suggested would ignore the significance all these words have, resulting in the loss of the unmistakable motif we’re so eager to find.

The last issue may come up with the solution of accepting some mismatches in the words we insert into our dictionary. We have to realize that this solution can rapidly increase the runtime of our procedure since for a word with length $k$, there are $\binom{k}{2} \cdot 3^2$ variants for 2 mismathes. This increases exponentially in the amount of mismatches we allow.
4.2 Heuristics

There is extensive literature concerning the issue of coming up with the word we need using heuristics. The underlying key element passing through all of them includes a systematic search for words.

To finalize our discussion it is best to briefly explore two such heuristics:

1. A first suggestion would be to extend our alphabet with the letter N in order to represent the any letter. (i.e. $\Sigma = \{A, C, T, G, N\}$). Thus, every time we stumble upon a word, we would first insert it into our dictionary as the original word as well as the word with the letter N instead of each other letter.

For example: Let’s assume that the word we encounter is:

$$ACTTGCTA$$

Then we would insert it into the dictionary but we would also insert:

$$NCTTGCTA$$
$$ANTTGCTA$$
$$ACNTTGCTA$$
$$\ldots$$
$$ACTTGCTN$$

This method enables us to exhaust our data and handle the problems mentioned before. It is also possible to extend the alphabet we choose as we wish.

2. The second suggestion includes an interesting probabilistic method: It is based on the fact that a protein is less likely to have changes in specific positions we can put a finger on. By guessing the locations of mismatches and considering only the words without these mismatches we can significantly improve the runtime. In other words, we would randomly pick the locations of the mismatches we allow by only taking subsets of words from the original words.

For example: Assume that the sequence we’re encountered with is:

$$ACTTGCTA$$

And that we randomly chose the 3 and 5 indices for possible mismatches. Then we would insert into our dictionary only the word:

$$ACTCTA$$

(removing the third ‘T’ and the fifth ‘G’).

Now, this solution depends mostly on luck. If we choose the exact locations which keep appear in the sequences and does not change, we’ve done it. Evidently, if we repeat this method a relatively small amount of times, we get good results compared to the non-probabilistic methods.

As mentioned before, these heuristics are carried out over a discrete alphabet and as a result allow to systematically cover all word options.

Generally, a combination of the two methods works best. The systematically search finds the words we look for in a discrete space. Then, the random selection can imply which are the most likely solutions and finally, from the EM algorithm shown above it is possible to extract a better profile.
5 Summary

We have thoroughly discussed the darkest details of how to implement simple HMM learning. We took advantage of the elemental traits of the HMM in order to develop an algorithm which uses EM in order to find a motif within given sequences. We also troubled ourselves with the runtime, convergence and starting points of this algorithm. Finally, we have referred to several methods for solving our problem which is based on randomness as well as on our profound intuition.