Bayesian Modelling

Study Notes by Lidia Yamamoto, on the Course by Prof. Yves Moreau, KULeuven

January 21, 2016

Contents

1 Part 1: Sequence Alignment **3**
1.1 Substitution Matrices . . . 3
1.1.1 PAM and BLOSUM Matrices . . . 4
1.2 Gap Penalties: Opening Gaps vs. Extension Gaps . . . 5
1.3 Shortest Path with Dynamic Programming . . . 5
1.4 Global Alignment: Needleman-Wunsch Algorithm . . . 5
1.5 Local Alignment: Smith-Waterman Algorithm . . . 8
1.6 Statistical Significance of Alignments . . . 8
1.6.1 Extreme Value Distribution (EVD) . . . 9
1.6.2 Assessing Scores Based on the EVD . . . 9
1.7 BLAST . . . 10

2 Part 2: Bayesian Statistics **12**
2.1 Data, Model and Parameters . . . 12
2.2 Product Rule, Bayes' Rule, and Marginalization . . . 12
2.3 Prior, Posterior, Evidence, Likelihood . . . 13
2.3.1 Relationship Between Prior, Posterior, Evidence, and Likelihood . . . 14
2.4 Maximum Likelihood (ML) (sl 16) . . . 14
2.5 Maximum a Posteriori (MAP) (sl 17) . . . 15
2.6 Posterior Mean Estimate (PME) (sl 18 lec 5) . . . 15
2.7 Bayesian Inference . . . 15
2.7.1 Examples of Bayesian Inference . . . 16
2.8 Likelihood Estimation Using the Multinomial Distribution . . . 17
2.8.1 Maximum Likehood Estimation of Frequency Matrices . . . 18
2.9 Prior and Posterior Estimations Using the Dirichlet Distribution . . . 20
2.9.1 Prior Estimation Using the Dirichlet Distribution . . . 20
2.9.2 Bayesian Inference of the Posterior Distribution . . . 21
2.9.3 Estimating the Average of the Posterior: Pseudocounts . . . 22
2.10 Dealing with Heterogeneity: The Dirichlet Mixture . . . 23

3 Part 3: Hidden Markov Models (HMM) **26**
3.1 Log Odds to Score Sequences . . . 26
3.2 Markov Chain . . . 26
3.2.1 Markov Property . . . 26
3.3 Hidden Markov Model (HMM) . . . 27
3.3.1 Markov Property in HMM . . . 28
3.4 Viterbi Algorithm . . . 29
3.5 Forward Algorithm . . . 30
3.6 Backward Algorithm . . . 31
3.7 Posterior decoding sl 20 . . . 32

3.8 Parameter estimation with known paths sl 25 . . . 32
3.9 Parameter estimation with unknown paths sl 27 . . . 33
3.9.1 Viterbi training . . . 33
3.9.2 Baum-Welch training sl 28 . . . 33
3.10 HMM Applications . . . 35
3.10.1 Profile HMMs . . . 35
3.10.2 Gene Finding . . . 35

4 Part 4: Expectation Maximization (EM) 36
4.1 Baum-Welch algorithm . . . 38
4.2 Motif finding . . . 38

1 Part 1: Sequence Alignment

The goal of sequence alignment is to find similarities between two sequences. These similarities might indicate homology, that is, highly similar sequences most likely share a common ancestor.

Alignments receive a score that measures how good they are. In order to score alignments substitution matrices such as PAM or BLOSUM are used in combination with gap penalties. When scoring alignments, lower numbers mean worse score, and higher numbers mean better score.

1.1 Substitution Matrices

It is important to know how to obtain the alignment using substitution matrices, and how to obtain the substitution matrix from data. The data here come from trusted alignments which are known to be true from biology.

The substitution score is obtained by looking up the substitution matrix $s(x, y)$ for symbols x_i and y_j. Scores reflect how often we actually see a given substitution: substitutions that more often encountered, hence more likely to occur biologically, receive a higher score; conversely, rare substitutions receive a lower score.

The diagonal of the substitution matrix scores a match in position (i, j) of the alignment, that is, $x_i = y_j$. The diagonal does not always contain the same numbers, because some matches are more better than others: for instance if in trusted alignments we see more A-A matches than C-C then the score of A-A in the diagonal of the substitution matrix will be higher than the score of C-C (meaning that seing an A-A match is more valuable than seing C-C).

In order to compute substitution scores we need to count the frequency of symbols (amino acids, nucleotides) and the frequency of their substitutions. Example for amino acids:

- q_Q: frequency of Q aminoacids
- p_{QR} frequency of Q $\leftrightarrow$ R substitutions

In general:

- q_{x_i}: frequency of symbol x_i in trusted alignments
- p_{x_i,y_j} frequency of $x_i \leftrightarrow y_j$ substitutions in trusted alignments

Consider two aligned sequences x and y after dropping the gaps in their alignment:

- Probability of x and y when they are both random:

$$P(x, y|R) = P(x|R)p(y|R) = q_{x_1} q_{x_2} ... q_{y_1} q_{y_2} ... = \prod_i q_{x_i} \prod_j q_{y_j} \tag{1}$$

- Probability of x and y when they are related:

$$P(x, y|M) = p_{x_1,y_1} p_{x_2,y_2} ... = \prod_i p_{x_i,y_i} \tag{2}$$

Compare equations (2) and (1) using the **odds ratio**, taking into account that after removing the gaps, the two sequences will have the same length:

$$\frac{P(x, y|M)}{P(x, y|R)} = \frac{\prod_i p_{x_i,y_i}}{\prod_i q_{x_i} \prod_j q_{y_j}} = \prod_i \frac{p_{x_i,y_i}}{q_{x_i} q_{y_i}} \tag{3}$$

If $P(x, y|M) << P(x, y|R)$ then x and y are unrelated, else they are related.

The differences are usually huge, moreover computing the product in eq. (3) can lead to numerical problems, so it is better to take the logarithm of the odds ratio, resulting in the **log-odds score**:

$$S(x,y) = \log\left(\frac{P(x,y|M)}{P(x,y|R)}\right) = \log\left(\prod_i \frac{p_{x_i,y_i}}{q_{x_i}q_{y_i}}\right) = \sum_i \log\left(\frac{p_{x_i,y_i}}{q_{x_i}q_{y_i}}\right) \tag{4}$$

The quantity $S(x,y)$ is the total score for the alignment. It is the sum of the individual scores $s(a,b)$ for each pair of aligned symbols, which are looked up in the substitution matrix:

$$s(a,b) = \log\left(\frac{p_{ab}}{q_a q_b}\right) \tag{5}$$

If two symbols are related (e.g. like the letters d-t, s-z, s-ss in Dutch vs. German), then the probability of seeing them together (p_{ab}) is higher than the probability of seeing them independently ($q_a q_b$):

- If a and b are **related** then $p_{ab} > q_a q_b \Rightarrow \log(p_{ab}/(q_a q_b)) > 0 \Rightarrow$ **positive score**
- If a and b are **unrelated** then $p_{ab} < q_a q_b \Rightarrow \log(p_{ab}/(q_a q_b)) < 0 \Rightarrow$ **negative score**

Hence the substitution score measures the probability that the two amino acids are related (that is, often aligned) in reality.

1.1.1 PAM and BLOSUM Matrices

Two frequently used substitution matrices are PAM and BLOSUM:

- Point Accepted Mutations (PAM):
  PAM1 matrix: 1% point accepted mutations
  PAM250 is 250% point accepted mutations ($\approx$ 20% similarity): PAM1 to the power 250.
  Beware that PAM doesn't work well for large evolutionary distances.
- BLOSUM matrix (BLOCKS SUbstitution):
  Computed from ungapped alignments in the BLOCKS database, which is a db of validated alignments.
  The number of the BLOSUM matrix (e.g. BLOSUM80, BLOSUM62, BLOSUM50) is the maximum percentage of similarity used in the construction of the matrix.

The PAM matrix tries to turn the evolutionary clock back in time by trying to bring sequences to the situation where there's only 1% differences between them. If two sequences are close (e.g. human vs chimp), the evolutionary clock will turn back only a little. Conversely, if two sequences are distant (e.g. human vs yeast), then the evolutionary clock must turn back a lot in order to get to the point where there was only 1 % difference between them.

A BLOSUM{L} matrix results from the comparison between sequences that have *at most* L% similarity: those sequences that have more than L% identical amino acids are grouped together in one cluster and considered as a single sequence: all the sequences that are sufficiently different (belonging to different clusters) are then aligned together and the score put in the BLOSUM{L} matrix.

A BLOSUM matrix with high L (e.g. BLOSUM80) is appropriate to align closely related sequences, such as mouse and human (with short evolutionary distance hence high similarity). Conversely, low L is suitable to align distant sequences such as human and yeast (low similarity). An intermediate matrix such as BLOSUM62 is appropriate to score sequences that are not so close, not so far, such as human and chicken.

1.2 Gap Penalties: Opening Gaps vs. Extension Gaps

The gap penalty penalizes for gaps in the alignment. It is typically a negative number: $G = -d < 0$ for a gap in one position. The lower the number the more severe the penalty (it makes the score of the alignment decrease).

Opening gap vs. **extension gap** penalty: sometimes a whole piece is missing in a sequence, but we don't want the penalty to be too big for it, just skip missing piece and go on. So the 2nd gap (extension gap) is usually cheaper (less severe penalty) than the 1st gap (opening gap):

```
e.g. -8 penalty for 1st gap (= opening gap)
     -4 penalty for 2nd gap (= extension gap)
```

1.3 Shortest Path with Dynamic Programming

Dynamic programming is used to solve the shortest path problem, e.g. finding shortest path between 2 cities (like GPS navigator), see 1. It is based on **Belman's optimality principle**: if I have the shortest path (optimal), then any subpath must be optimal too: there cannot be a shortcut in the optimal path, otherwise it wouldn't be the optimum!

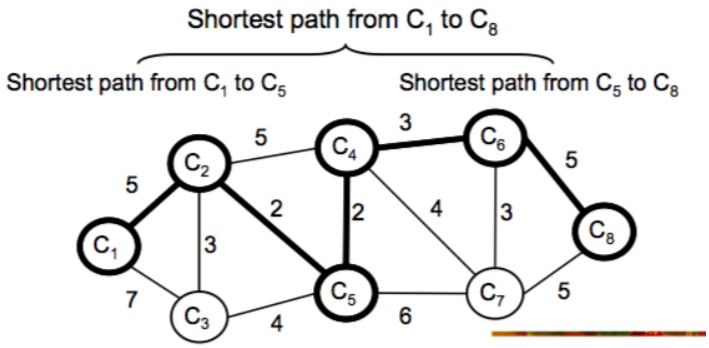

Figure 1: Solving the shortest path problem

For instance, in figure 1, either C6 or C7 is in the shortest path to C8. So if I know the shortest path to C6 or to C7, then I know which one is on the shortest path to C8.

```
e.g. if C1-C6 = 12
        C1-C7 = 13
      then C1-C6-C8 = 12+5 = 17 better (choose this one)
           C1-C7-C8 = 13+5 = 18 worse (discard)
best way to reach C6:
C1-C4 = 9 ==> C1-C4-C6 = 9 + 3 = 12
C1-C5 = 7 ==> C1-C5-C7 = 7 + 6 = 13
etc.
```

1.4 Global Alignment: Needleman-Wunsch Algorithm

Given two sequences $x = x_1, ..., x_i, ...$ and $y = y_1, ..., y_j, ...$ to align (or *query* and *target*), make a matrix F of the alignment, with one sequence in the rows and the other in the columns, plus one starting position (0,0) at the top left corner. Initially the matrix is empty, as shown in figure 2. We will fill in the matrix F such that position (i, j) gives the score $F(i, j)$ of the best alignent up to this position (with higher numbers meaning better score).

IMPORTANT!! The convention here is that i is the **column** and j is the **row** of matrix F !!! Hence the first sequence x is placed horizontally (one symbol per column), and the second sequence y is placed vertically (one symbol per row).

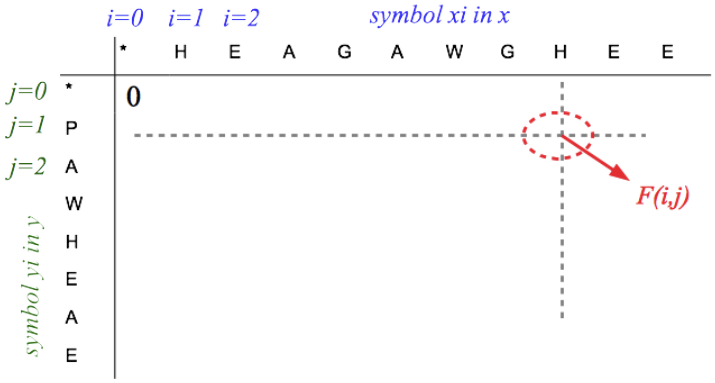

Figure 2: Initial state of matrix F of the alignment.

An alignment can be seen as a particular path through the matrix of the alignment. The problem of finding the best alignment is then the problem of finding the legal path with the highest score, using only 3 legal movements through the matrix: diagonally down, right or down. This is analogous to the shortest path problem, and can be solved by dynamic programming.

Legal movements:

- Diagonally down = from $(i-1, j-1)$ to (i, j): using substitution score $S(x_i, y_j)$
- Right or Down using gap score:
  - Vertically **down** = from $j-1$ to j: **insertion** in the second strand
    we move down, hence in the direction of the second strand (x) whereas the 1st strand (x) gets stalled (hence we put a gap in the first one, so there's an insertion in the second one)
  - Horizontally to the **right** = from $i-1$ to i: **deletion** in the second strand
    we move right, hence in the direction of the first strand (x) whereas the 2nd strand (y) gets stalled (hence we put a gap there, meaning that the symbol is deleted from the 2nd one)

How to fill in the matrix: start from the top left corner $F(0,0) = 0$, then complete each column successively computing the alignment score $F(i,j)$ as the best of 3 possible scores:

Substitution:	$S = F(i-1,\ j-1) + s(x_i, y_j)$	move diagonally using subst. score $s()$
Insertion:	$D = F(i,\ j-1) - d$	move down using gap penalty: $G = -d$
Deletion:	$I = F(i-1,\ j) - d$	move right using gap penalty
Score:	$F(i,j) = \max(S, I, D)$	maximum of the 3 possibilities above

This is shown in Figure 3.

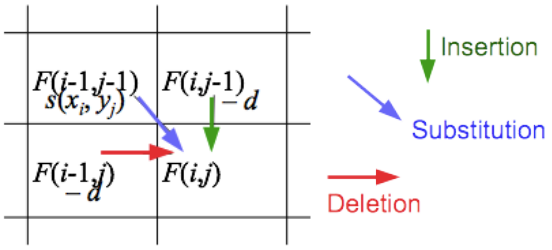

Figure 3: How to fill in the matrix F of the alignment.

An example of resulting matrix F after filling in all the positions is shown in figure 4

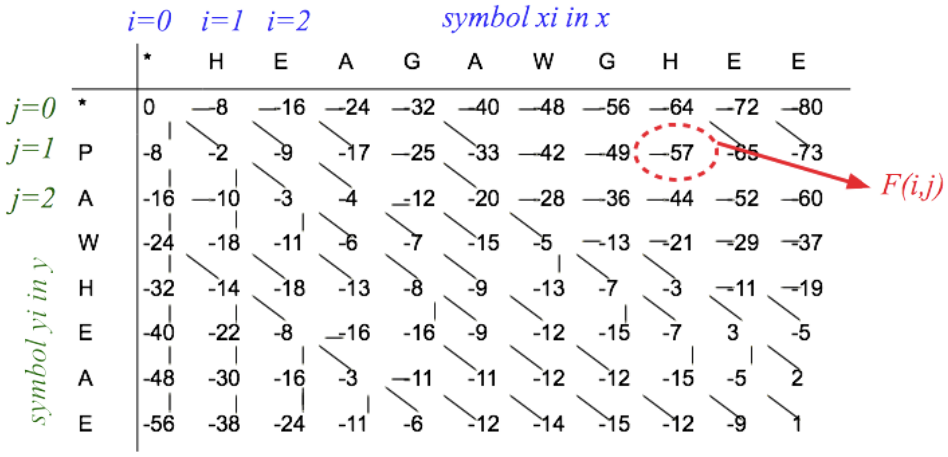

Figure 4: Example of filled matrix F.

An alignment can be seen as a particular path through the matrix, following the best path found in the maximization procedure above. In order to determine the best global alignment, look at the last position of F (bottom right slot), and retrace the maximization steps back to the first position $(0, 0)$, building position k of the alignment a_x, a_y at each step:

- if the optimum at position (i, j) was a substitution, put align symbol x_i and symbol y_j on top of each other, and move to position $(i - 1, j - 1)$.
- if the optimum was an insertion (vertical bar) then a_{xk} = '-' (gap) and $a_{yk} = y_j$; move upwards to position $(i, j - 1)$.
- if the optimum was a deletion (horizontal bar) then $a_{xk} = x_i$ a_{yk} = '-' (gap); move left to position $(i - 1, j)$.
- Decrement k and repeat this until we reach $F(0, 0) = 0$.

Figure 5 shows an example of global alignment resulting from this procedure.
Note that the gap scores of this matrix do not match those of figure 4. Probably an extension penalty of zero was applied!?!?!?

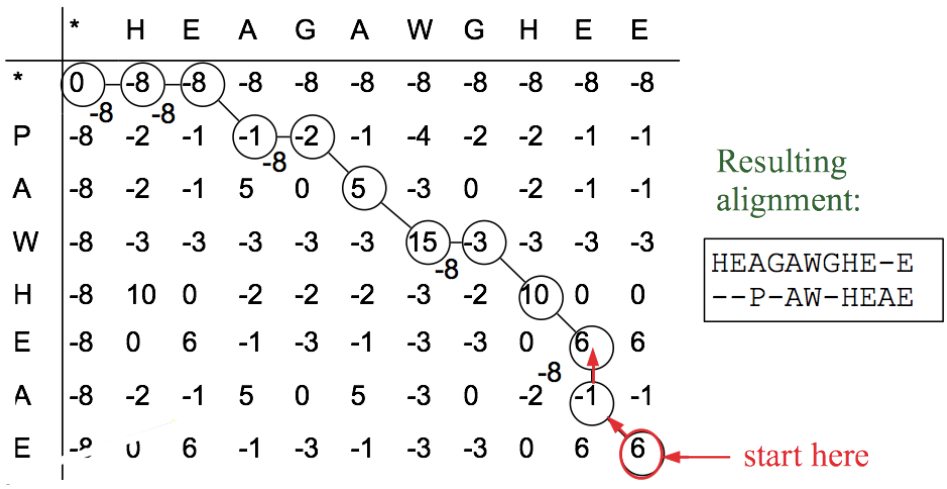

Figure 5: Finding the optimum global alignment by tracing matrix F from the bottom-right position back to $(0, 0)$.

1.5 Local Alignment: Smith-Waterman Algorithm

The goal of a local alignment algorithm is to find the best matching subsequence between two sequences. The algorithm is similar to the global alignment, with two phases: fill in the F matrix one by one (from left-top to right-down), then traceback following the pointers to the maximum scores. Except that now negative scores get reset to zero when filling in F, and we start from the highest score (and not from the bottom-right position) when tracing back.

This is the procedure to fill in the F matrix:

Initialization:	$F(0,0) = 0$	
Substitution:	$S = F(i-1,\ j-1) + s(x_i, y_j)$	move diagonally using subst. score $s()$
Insertion:	$D = F(i,\ j-1) - d$	move down using gap penalty: $G = -d$
Deletion:	$I = F(i-1,\ j) - d$	move right using gap penalty
Restart:	$R = 0$	
Score:	$F(i,j) = \max(S, I, D, R)$	maximum of the 4 possibilities above

The procedure to trace back and find the local alignment is shown in figure 6 using an example.

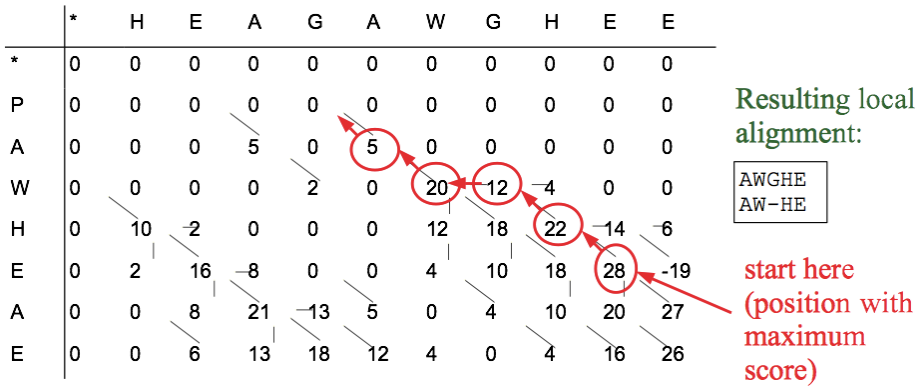

Figure 6: Example of finding the optimum local alignment by tracing back matrix F from the position with highest score back to the position where the score becomes zero.

Both local and global alignment algorithms have quadratic complexity.

1.6 Statistical Significance of Alignments

Once we have a score for our alignment, how to judge if this score good or bad? How to assess if an alignment is good or bad? Answer: we must assess its statistical signifcance, hence compute its p-value. A small p-value means that the alignment is significantly better than what can be expected by random chance.

The computation of the p-value of an alignment relies on the fact that the score of a pair random sequences is normally distributed: if we pick two random sequences, align them, get the score, then repeat this many many times, the set of scores obtained is normally distributed. So a statistically significant score must do a lot better than this normal distribution. How much better? Better than most of the best random scores. For this we must look at the distribution of maximum scores out of a set of random scores.

The maximum of a set of normally distributed values (scores of random sequences in our case) follows the Extreme Value Distribution (EVD). A significantly good alignment must have a score that is much larger than most of the best random scores, which are EVD distributed. That is, the score for a good alignment should lie at the extreme right tail of the EVD distribution. This is explained in more detail below.

1.6.1 Extreme Value Distribution (EVD)

If I repeat the following procedure several times:

- draw n samples from a normal distribution
- take the maximum M_n of these n samples

then the probability that the maximum M_n is within a specified value x follows the **extreme value distribution (EVD)**:

$$P(M_N \leq x) = \exp(-K\ n\ e^{\lambda(x-\mu)}) \tag{6}$$

this is an exponential of an exponential!!

Figure 7 shows the shape of the unscaled EVD, for $\lambda = 1$, $\mu = 0$, and $Kn = 1$.

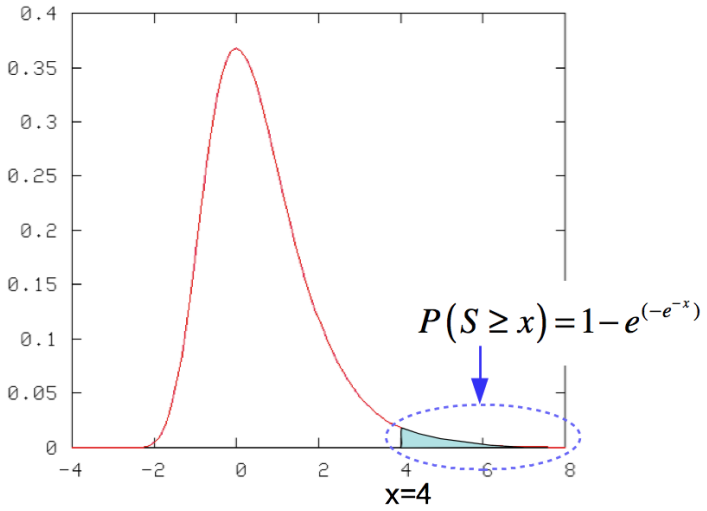

Figure 7: Extreme Value Distribution, unscaled. For an alignment to be significant, the shaded area under the curve should be sufficiently small. From `http://elbo.gs.washington.edu/courses/GS_373_12_sp/slides/6A-significance_scores.pdf`

For ungapped alignments, the EVD has the form of eq. (6). For gapped alignments, the EVD has the extended form:

$$P(S \leq x) = \exp(-K\ n\ m\ e^{-\lambda x}) \tag{7}$$

where n is the length of the query, and m is sum of the lengths of all sequences in the target database.

1.6.2 Assessing Scores Based on the EVD

How high my score has to get to make sure it doesn't come from the normal distribution of random scores? It must be better than the best random alignments among a large number of sequences.

Let x be the score of my alignment. If the probability of getting a maximum score S better than my score x is sufficiently small, say, less than 1%, then my alignment is significant:

If $P(S > x) = 1 - P(S \leq x) < 0.01$ then the alignment is significant.
where $P(S \leq x)$ is calculated according to eq. (7).

1.7 BLAST

BLAST is used to search for a sequence in a database of sequences. The sequence we're looking for is called the *query* and the sequences stored in the database are the *targets*. Due to the huge number of target sequences, Smith-Waterman local alignment would be too slow, so a more efficient algorithm is needed.

Basic Local Alignment Search Tool (BLAST) solves this problem using heuristics. It works by performing a fast and cheap search at the beginning, that allows us to eliminate most of the non-promising targets, and then focusing on the fewer potentially promising hits for closer inspection.

Steps of the BLAST algorithm:

- **Step 1: look for triplets**
  This step is cheap, computationally. Given a query such as 'RNQHEEWAPRDCNRH':
  - look up all the triplets in this sequence: RNQ, NQH, QHE, ..., etc. in potential targets;
  - also include triplets that are not in the query but are similar enough:
    RNQ: RNE, RNH, RQQ ...
    NQH: NHQ, NHA, NRH, ...
  - make a tree out of all the triplets:

```
R -> N -> Q
       -> E
       -> H
  -> Q -> Q
       -> E
N -> R -> H
       -> Q
  -> Q -> W
       -> A
       -> H
```

- **Step 2: regular expression search**
  - Use a regular expression algorithm (finite state automaton) to perform the search for the triplets in the db.
  - Keep track of the match position in query and target.
  - The score doesn't matter much at this point, and doesn't depend on the number of matching triplets: if the same triplet is found several times, or multiple triples found in targets, the score is not higher, just mark all the targets that match.

- **Step 3: select pairs of closeby hits**
  - take 2 triplet hits next to each other (or very close):
    assume subblock without gaps with 2 matching hits, in order to avoid dynamic programming (this might miss a good alignment but with lots of gaps, but we hope it doesn't occur too ofen)
  - discard targets with no 2 hits

- **Step 4: extend without gaps**
  - for each block found in step 3, try to extend the block in both directions (left and right), see if still matches nicely:

```
NQHEEWAPRDCNRH
NRHWWWRPRNCANR
       ++-+---

+ : increase score
- : decrease score
```

  - score goes up with a match, down with mismatch (remember that no gaps are considered at this stage)
  - keep the highest score that you've been able to extend without gaps (*high scoring segment pair* (**HSP**))
  - extend further until the score decreases too much

- **Step 5: extend HSPs with gaps**
  - extend the HSPs by gapped alignment
  - stop the extension when the score drops by X_g (e.g., X_g=40) under the best score so far;
  - select the best gapped local alignment

- **Step 6: assess significance**
  - compute the statistical significance of the alignments (hits);
  - for the significant alignments, repeat the gapped alignment (step 5) with a higher dropoff parameter X_g for more accuracy.

2 Part 2: Bayesian Statistics

2.1 Data, Model and Parameters

We would like to build a model of the world M with parameters θ based on some observed data D. The model should be such that it is possible to tune its parameters θ until they fit the data D as closely as possible, so that the model with the parameters closely reproduce the reality.

We focus on biological problems typically involving the analysis of DNA and protein sequences, so here are some examples of the kind of data, parameters and models we'll be looking at:

- Problem: Estimate natural frequencies of nucleotides or amino acids based on data from multiple sequence alignments:
  - **Data D:** A set of counts n for each type of nucleotide or amino acid in a column of a multiple sequence alignment.
  - **Model M:** Maximum likelihood and posterior mean estimate using multinomial and Dirichlet distributions, with a Dirichlet mixture in case multiple classes (*sources*) are present.
  - **Parameters θ:** The estimated frequency $\theta_k = q_k$ of each type of nucleotide or amino acid in a column of a multiple sequence alignment. The frequencies may apply to the whole alignment or may be divided into classes $S = k$ (sources) in case regions of the alignment have different characteristics leading to different frequency distributions (e.g. hydrophobic vs. hydrophilic regions). The path π through a set of states in a HMM.
- Problem: Look for patterns in a given sequence or set of sequences:
  - **Data D:** A set of sequences x each of length L (variable or fixed length).
  - **Model M:** HMM with a predefined architecture (set of states and transitions).
  - **Parameters θ:** The transition probabilities a_{kl}, emission probabilities $e_k(b)$, and the path π through a set of states in the HMM.

2.2 Product Rule, Bayes' Rule, and Marginalization

- **Product rule:** (sl 9, lec 4)
  The probability that A and B are true at the same time given C is equal to the probability of B given A times the probability of A.

$$\begin{aligned} P(A,B|C) &= P(B|A,C) * P(A|C) = \\ P(B,A|C) &= P(A|B,C) * P(B|C) \end{aligned} \tag{8}$$

  When C is always given anyway, we can simplify the formula by making C implicit:

$$\begin{aligned} P(A,B) &= P(B|A) * P(A) = \\ P(B,A) &= P(A|B) * P(B) \end{aligned} \tag{9}$$

- **Bayes' rule:** (sl 9, lec 4)

  Rearranging eq. (8) leads to two equivalent forms of Bayes' Rule:

$$P(B|A,C) * P(A|C) = P(A|B,C) * P(B|C) \Rightarrow$$

$$\boxed{P(B|A,C) = \frac{P(A|B,C)P(B|C)}{P(A|C)}} \tag{10}$$

or equivalently:

$$\boxed{P(A|B,C) = \frac{P(B|A,C)P(A|C)}{P(B|C)}} \tag{11}$$

Starting from equation (9), we obtain the more popular form of Bayes rule:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{12}$$

- **Marginalization rule**, or marginalization trick: (sl 22, lec 4)
  The sum of all the possible outcomes on one particular dimension of the problem must add up to 1, that is, the probabilities of all the possible cases must add up to one.

$$\sum_{k=1}^{K} P(Y=k) = 1 \tag{13}$$

$$\sum_{k=1}^{K} P(X, Y=k) = P(X) \tag{14}$$

$$\int_{y \in Y} P(X, Y=y)dy = P(X) \tag{15}$$

  We often encounter a conditional probability case where a hidden variable Y has an influence on $P(A|B)$, but we cannot assume any particular value for Y. In this case, in order to compute $P(A|B)$ we must take into account all possible values of Y given B, weighted by their respective probabilities:

  – For Y discrete:

$$\begin{aligned} P(A|B) &= \sum_{k=1}^{K} P(A, Y=k|B) \\ &= \sum_{k=1}^{K} P(A|Y=k, B)\ P(Y=k|B) \end{aligned} \tag{16}$$

$$\tag{17}$$

  – For Y continuous:

$$\begin{aligned} P(A|B) &= \int_{y \in Y} P(A, Y=y|B)\ dy \\ &= \int_{y \in Y} P(A|Y=y, B)\ P(Y=y|B)\ dy \end{aligned} \tag{18}$$

  This conditional form of the marginalization rule is often used.

2.3 Prior, Posterior, Evidence, Likelihood

- $P(\theta|M) =$ **Prior**: A priori knowledge or *belief* about what the parameters θ of my model M might be, *before* we actually see any data D. The prior is a distribution of the parameters θ. The initial prior should be as neutral as possible: avoid zero!

- $P(D|\theta, M)$ = **Likelihood** of the data: Probability of seeing the data D with the model M under the model parameters θ, *before* we actually see any data D: for a given combination of $\{\theta, M\}$, we'll see data D of a certain form, for another combination of $\{\theta, M\}$, we'll see another kind of data D.

  We will usually be interested in the **maximum likelihood** that is, the combination of θ parameters for a given model M that maximizes the likelihood of seeing D, without assuming that D is given (section 2.4).

- $P(\theta|D, M)$ = **Posterior**: Updated probability of parameters θ *after* seeing the data D. Usually more peaked than the prior $P(\theta|M)$ because now we know more. We reduced the uncertainty by seeing D, hence reduced the standard deviation σ, making the probability density function narrower.

- $P(D|M)$ = **Evidence**: Probability of observing the data D with the given model M, independently of the parameters θ used. Therefore to compute the evidence we must take into account all possible values that the parameters θ can take to produce D from M, that is: the weighted sum of the probability of each possible set of parameters ($P(\theta|M)$, the prior) weighted by the chance of observing D with the model M under that particular set of parameters ($P(D|\theta, M)$, which is the likelihood). This can be computed by marginalization:

$$P(D|M) = \int_\theta P(D|\theta, M)P(\theta|M)d\theta \tag{19}$$

  when θ is continuous, or

$$P(D|M) = \sum_\theta P(D|\theta, M)P(\theta|M) \tag{20}$$

  when θ is discrete.

2.3.1 Relationship Between Prior, Posterior, Evidence, and Likelihood

Using Bayes' rule:

$$P(\theta|D, M) = P(D|\theta, M)\,\frac{P(\theta|M)}{P(D|M)} \Rightarrow \tag{21}$$

$$P(\theta|D, M)\;P(D|M) = P(D|\theta, M)\;P(\theta|M) \tag{22}$$

which means:

$$\text{posterior} * \text{evidence} = \text{likelihood} * \text{prior} \tag{23}$$

2.4 Maximum Likelihood (ML) (sl 16)

Seek the values of θ that give the maximum probability of seeing the data with the given model, *without knowing the data in advance!!*
Often expressed in logarithmic scale:

$$\boxed{\theta^{\text{ML}} = \underset{\theta}{\text{argmax}}\;\; P(D|\theta, M) = \underset{\theta}{\text{argmax}}\;\; \ln\; P(D|\theta, M)} \tag{24}$$

Tweek θ_j until the probability $P(D|\theta, M)$ becomes maximum.

2.5 Maximum a Posteriori (MAP) (sl 17)

Pick value of θ such that the probability of θ *given the data and the model* is maximum: so here you *have* the data, in contrast with ML.

$$\theta^{MAP} = \text{argmax } P(\theta|D, M) \tag{25}$$

Using Bayes' rule:

$$P(\theta|D, M) = P(D|\theta, M)\frac{P(\theta|M)}{P(D|M)} \Rightarrow \tag{26}$$

$$\theta^{MAP} = \text{argmax } P(\theta|D, M) = \text{argmax } P(D|\theta, M)\frac{P(\theta|M)}{P(D|M)} \tag{27}$$

The evidence term $P(D|M)$ plays no role in the maximization over θ, so it can be left out in the computation of θ^{MAP}:

$$\boxed{\theta^{\text{MAP}} = \text{argmax } P(\theta|D, M) = \text{argmax } P(D|\theta, M)P(\theta|M)} \tag{28}$$

In other words:

$$\boxed{\theta^{\text{MAP}} = \text{argmax(posterior)} = \text{argmax(likelihood} * \text{prior)}} \tag{29}$$

2.6 Posterior Mean Estimate (PME) (sl 18 lec 5)

The Posterior Mean Estimate (PME) is the average of the posterior distribution $P(\theta|D, M)$ over all possible values of θ:

$$\theta^{\text{PME}} = \int_{\theta \in H} \theta \; P(\theta|D, M) \; d\theta \tag{30}$$

2.7 Bayesian Inference

In contrast with ML and MAP which only take the maximum, in Bayesian inference we compute whole posterior distribution $P(\theta|D, M)$. Using Bayes rule for the posterior (eq. (21)):

$$\boxed{P(\theta|D, M) = P(D|\theta, M) \; \frac{P(\theta|M)}{P(D|M)}} \tag{31}$$

Which is the same as:

$$\text{posterior} = \text{likelihood} * \text{prior} \; / \; \text{evidence} \tag{32}$$

Applying the marginalization rule for the evidence (eq. (19)):

$$P(\theta|D, M) = P(D|\theta, M) \; \frac{P(\theta|M)}{P(D|M)} \Rightarrow \tag{33}$$

$$P(\theta|D, M) = \frac{P(D|\theta, M) \; P(\theta|M)}{\int_{w \in \theta} P(D|w, M) \; P(w|M) \; dw} \tag{34}$$

where the integral $\int_w$ sums over all possible values of θ.

If θ is discrete:

$$P(\theta = i|D, M) = \frac{P(D|\theta = i, M) \; P(\theta = i|M)}{\sum_{j=1}^{K} P(D|\theta = j, M) \; P(\theta = j|M)} \tag{35}$$

Another way to arrive at eq. (34) from eq. (33) is to reason that, since we don't have the evidence $P(D|M)$, we can consider that $P(\theta|D,M)$ is proportional to $P(D|\theta,M)P(\theta|M)$: the integral of this product will not sum to one because the term $P(D|M)$ is missing:

$$\int_\theta P(\theta|D,M)\ d\theta = 1 \Rightarrow \tag{36}$$

$$\int_\theta P(D|\theta,M)\ \frac{P(\theta|M)}{P(D|M)}\ d\theta = 1 \Rightarrow \tag{37}$$

$$\frac{1}{P(D|M)}\int_\theta P(D|\theta,M)\ P(\theta|M)\ d\theta = 1 \Rightarrow \tag{38}$$

$$\int_\theta P(D|\theta,M)\ P(\theta|M)\ d\theta = P(D|M) \tag{39}$$

which results in the marginalization rule in eq. (18).

By substituting eq. (39) in eq. (33) we get eq. (34) again as expected (after a change of variables to distinguish the variable θ from all its possible values w):

$$P(\theta|D,M) = \frac{P(D|M,\theta)\ P(\theta|M)}{\int_w P(D|w,M)\ P(w|M)\ dw} \tag{40}$$

Rewriting the marginalization trick for the evidence $P(D|M)$:

$$P(D|M) = \int_\theta P(D,\theta|M)d\theta \tag{41}$$

$$= \int_\theta P(D|\theta,M)P(\theta|M)d\theta \tag{42}$$

2.7.1 Examples of Bayesian Inference

- Diagnostic test for HIV: is patient carrier of HIV virus?

  Random variable $H \in \{H+, H-\}$: $H+$ = patient is carrier; $H-$ = patient is not carrier.

  Blood sample test T returns positive ($T+$) or negative ($T-$): $T \in \{T+, T-\}$.

  The test can return false positives or false negatives, which should be minimized.

  Given that if a patient has HIV, then the probability of the test returning $T+$ is:
  $P(T+|H+) = 99.8\%$.
  Whereas if the patient is not infected, the probability that the test returns $T+$ is:
  $P(T+|H-) = 0.9\%$.

  These numbers are given and fixed for this test. Another parameter that is given is the proportion of infected people in the population. For instance, $P(H+) = 0.002$ in Belgium, $P(H+) = 0.2$ in another country with much larger infected population.

  Typical test in Belgium:

  $$\begin{aligned} P(T+|H+) &= 0.998 \text{ given} \\ P(T+|H-) &= 0.009 \text{ given} \\ P(H+) &= 0.002 \text{ given} \end{aligned}$$

  Same test in a country with a much larger infected population:

  $$\begin{aligned} \text{P(T+—H+)} &= 0.998 \text{ given} \\ \text{p(T+—H-)} &= 0.009 \text{ given} \\ \text{p(H+)} &= 0.2 \text{ given} \end{aligned}$$

If a patient tests $T+$, what is the probability that s/he has HIV?
Using Bayes' rule:

$$P(H+|T+) = P(T+|H+)\ \frac{P(H+)}{P(T+)} \tag{43}$$

Use marginalization: the probability of $T+$ is the probability of testing positive and being HIV+ or being HIV-:

$$\begin{aligned} P(T+) &= P(T+,H+) + P(T+,H-) && (44)\\ &= P(T+|H+)\ P(H+) + P(T+|H-)\ P(H-) && (45)\\ P(H-) &= 1 - P(H+) && (46) \end{aligned}$$

$P(H+|T+)$ in Belgium:

$$\begin{aligned} P(H+|T+) &= P(T+|H+)\ P(H+) + P(T+|H-)\ P(H-) && (47)\\ &= 0.988 * 0.002/(0.988 * 0.002 + 0.009 * 0.998) && (48)\\ &= 0.1803248768 \approx 18\% && (49) \end{aligned}$$

$P(H+|T+)$ in very infected country:

$$\begin{aligned} P(H+|T+) &= 0.988 * 0.2/(0.988 * 0.2 + 0.009 * 0.8) && (50)\\ &= 0.96484375 \approx 96\% && (51) \end{aligned}$$

The larger infected population means that there is a high chance of actually being infected after testing positive, whereas in a small infected population the chance of actually having the disease after testing positive is much lower, because there are so many less people infected in absolute terms.

- Is the Casino cheating?
  Coin: $\theta = 0$ or $\theta = 1$
  $P(\theta = i|M)$ is the prior probability of casino cheating
  ...
- Multiple Sequence Alignment
  ...

2.8 Likelihood Estimation Using the Multinomial Distribution

Say that we are computing a multiple sequence alignment, and get, for each column of the alignment, a given set of counts for each amino acid or base. These counts are my data D. For one particular column of the alignment, we get counts $\vec{n} = n_1, ...n_K$. These counts sum to N:

$$N = n_1 + n_2 + ... + n_K = \sum_{i=1}^{K} n_i \tag{52}$$

Given the natural frequencies $\theta_i, i = 1, ..., K$ of each amino acid or base that we are aligning (where K=4 for DNA sequences, K=20 for protein sequences), we would like to know how likely our observed counts n_i are. Our goal is to estimate the likelihood:

$$P(D|\theta, M) = P(\vec{n}|\vec{\theta}) = P(n|\theta) = P(N_1 = n_1, N_2 = n_2, ..., N_K = n_K|\theta_1, ..., \theta_K) \tag{53}$$

Note that the explicit vector indicators are often dropped for simplicity, but we must not forget that $n = \vec{n}$ and $\theta = \vec{\theta}$ are actually vectors.

This probability follows a Multinomial Distribution:

$$P(n|\theta) \sim \mathcal{M}(n;\theta) = \frac{1}{M(n)} \prod_{i=1}^{K} \theta_i^{n_i} \tag{54}$$

where $M(n)$ is given by:

$$M(n) = \frac{\prod_{i=1}^{K} n_i!}{N!} \tag{55}$$

For K=2, the multinomial distribution becomes the binomial distribution.

2.8.1 Maximum Likehood Estimation of Frequency Matrices

Given a count vector of observations $\vec{n} = n_1, ...n_K$, how to estimate the frequencies $\vec{\theta} = \theta_1, ..., \theta_K$ that best explain these observations? Intuitively the answer is $\theta_i \approx n_i/N$. We now demonstrate that this intuitive result actually follows from a maximum likelihood estimation of θ given n, using the fact that $P(n|\theta)$ is multinomially distributed.

The goal is to solve the maximization problem:

$$\theta^{ML} = \underset{\theta}{\operatorname{argmax}}\, P(n|\theta) = \underset{\theta}{\operatorname{argmax}}\, \mathcal{M}(n;\theta) \tag{56}$$

under the constraint that $\sum_{i=1}^{K} \theta_i = 1$ (there is also the constraint that $0 \leq \theta_i \leq 1 \forall i$ but let's put it aside it for now). The formulation of the optimization problem becomes:

$$\max f(\theta) = \mathcal{M}(n;\theta) \quad \text{subject to} \tag{57}$$

$$g(\theta) = \sum_{i=1}^{K} \theta_i - 1 = 0 \tag{58}$$

This problem can be solved using the method of Lagrange multipliers:

$$\max \mathcal{Q}(\theta, \lambda) = f(\theta) + \lambda g(\theta) = \mathcal{M}(n;\theta) + \lambda \left(\sum_{i=1}^{K} \theta_i - 1 \right) \tag{59}$$

The solution is found by setting both partial derivatives of $\mathcal{Q}$ to zero:

$$\frac{\partial \mathcal{Q}(\theta, \lambda)}{\partial \theta} = 0 \quad \text{and} \tag{60}$$

$$\frac{\partial \mathcal{Q}(\theta, \lambda)}{\partial \lambda} = 0 \tag{61}$$

Solving eq. (61) simply leads to eq. (58), confirming that solving the unconstrained problem in eq. (59) is sufficient to solve the original constrained problem.

Let us solve eq. (60) for each θ_i:

$$\frac{\partial \mathcal{Q}(\theta, \lambda)}{\partial \theta_i} = \frac{\partial}{\partial \theta_i} \left(\mathcal{M}(n;\theta) \right) + \frac{\partial}{\partial \theta_i} \lambda \left(\sum_{i=1}^{K} \theta_i - 1 \right) \tag{62}$$

The second term of the above sum is easier to solve: all terms that do not involve θ_i become constants, with derivative zero. We're left with only the derivative of $\lambda \theta_i$ with respect to θ_i, which is simply λ:

$$\frac{\partial}{\partial \theta_i} \lambda \left(\sum_{i=1}^{K} \theta_i - 1 \right) = \lambda \tag{63}$$

Now let us solve the first term of eq. (62):

$$\frac{\partial}{\partial\theta_i}\left(\mathcal{M}(n;\theta)\right) = \frac{\partial}{\partial\theta_i}\left(\frac{1}{M(n)}\prod_{i=1}^{K}\theta_i^{n_i}\right) = \frac{1}{M(n)}\,\frac{\partial}{\partial\theta_i}\prod_{i=1}^{K}\theta_i^{n_i} \tag{64}$$

$$= \frac{1}{M(n)}\,\frac{\partial}{\partial\theta_i}\theta_1^{n_1}\times ...\times\theta_i^{n_i}\times ...\theta_K^{n_K} \tag{65}$$

$$= \frac{1}{M(n)}\,n_i\,\theta_1^{n_1}\times ...\times\theta_i^{n_i-1}\times ...\theta_K^{n_K} \tag{66}$$

$$= \frac{1}{M(n)}\,n_i\,\theta_1^{n_1}\times ...\times\frac{\theta_i^{n_i}}{\theta_i}\times ...\theta_K^{n_K} \tag{67}$$

$$= \frac{1}{M(n)}\,\frac{n_i}{\theta_i}\,\theta_1^{n_1}\times ...\times\theta_i^{n_i}\times ...\theta_K^{n_K} \tag{68}$$

$$= \frac{1}{M(n)}\,\frac{n_i}{\theta_i}\,\prod_{i=1}^{K}\theta_i^{n_i} \tag{69}$$

$$= \frac{n_i}{\theta_i}\,\frac{1}{M(n)}\,\prod_{i=1}^{K}\theta_i^{n_i} \tag{70}$$

$$= \frac{n_i}{\theta_i}\,\mathcal{M}(n;\theta) \tag{71}$$

Now combine eqs. (71) and (63) to obtain the solution of eq. (60):

$$\frac{\partial\mathcal{Q}(\theta,\lambda)}{\partial\theta} = 0\Rightarrow \tag{72}$$

$$\frac{n_i}{\theta_i}\,\mathcal{M}(n;\theta)+\lambda = 0 \tag{73}$$

The sum of eq. (73) over all i's should still be zero:

$$\sum_{i=1}^{K}\left(\frac{n_i}{\theta_i}\,\mathcal{M}(n;\theta)+\lambda\right) = 0\Rightarrow \tag{74}$$

$$\sum_{i=1}^{K}\left(\frac{n_i}{\theta_i}\,\mathcal{M}(n;\theta)\right)+\sum_{i=1}^{K}\lambda = 0\Rightarrow \tag{75}$$

$$\mathcal{M}(n;\theta)\sum_{i=1}^{K}\frac{n_i}{\theta_i}+K\lambda = 0 \tag{76}$$

Let's call $S=\sum_{i=1}^{K}\frac{n_i}{\theta_i}$:

$$\mathcal{M}(n;\theta)\;S+K\lambda=0\Rightarrow\lambda=-\frac{S}{K}\,\mathcal{M}(n;\theta) \tag{77}$$

Now substitute eq. (77) back in eq. (73):

$$\frac{n_i}{\theta_i}\,\mathcal{M}(n;\theta)+\lambda = 0\Rightarrow \tag{78}$$

$$\frac{n_i}{\theta_i}\,\mathcal{M}(n;\theta)-\frac{S}{K}\,\mathcal{M}(n;\theta) = 0\Rightarrow \tag{79}$$

$$\mathcal{M}(n;\theta)\left(\frac{n_i}{\theta_i}\;-\frac{S}{K}\right) = 0 \tag{80}$$

Assuming that $\mathcal{M}(n;\theta)\neq 0$, we must have:

$$\frac{n_i}{\theta_i}\;-\frac{S}{K}=0\Rightarrow\frac{n_i}{\theta_i}=\frac{S}{K}\Rightarrow\theta_i=\frac{K}{S}n_i \tag{81}$$

But remember our constraint that all θ_i's must sum to one, therefore:

$$\sum_{i=1}^{K} \theta_i = \sum_{i=1}^{K} \frac{K}{S} n_i = 1 \Rightarrow \frac{K}{S} \sum_{i=1}^{K} n_i = 1 \Rightarrow \frac{K}{S} N = 1 \Rightarrow S = KN \tag{82}$$

Now substitute eq. (82) back in eq. (81):

$$\theta_i = \frac{K}{S} n_i \Rightarrow \theta_i = \frac{K}{KN} n_i \Rightarrow \theta_i = \boxed{\theta_i^{\mathrm{ML}} = \frac{n_i}{N}} \tag{83}$$

which is the intuitive estimation of θ_i that is used everywhere. However there is a major problem with eq. (83): If there are no observations for amino acid or base i, hence $n_i = 0$, θ_i becomes zero which is not a good estimation unless we have a huge number of observations in total (huge N) so that we are sure that i really never occurs.

2.9 Prior and Posterior Estimations Using the Dirichlet Distribution

In the previous section we saw that the likelihood term of Bayesian inference follows a multinomial distribution. In this section we will see that the prior follows a Dirichlet distribution, and that the posterior can be inferred from the prior and the likelihood using Bayesian inference, resulting in a posterior that is also Dirichlet-distributed. The property that a Multinomial likelihood combined with a Dirichlet prior results in a Dirichlet posterior is called **conjugacy**:

```
posterior = likelihood  * prior     / evidence
    |           |           |
Dirichlet = Multinomial * Dirichlet / evidence
```

2.9.1 Prior Estimation Using the Dirichlet Distribution

When we don't know the frequencies $\theta_i, i = 1, ..., K$ we need to estimate the distribution of these frequencies. This is my prior distribution. The parameters $\vec{\theta} = \theta_1, ...\theta_K$ are probabilities, hence we are looking for the probability distribution of a vector of probabilities, which is like a *"dice factory"*. The Dirichlet distribution does this job, hence **the prior follows a Dirichlet Distribution**:

$$P(\vec{\theta}) \sim \mathcal{D}(\theta; \alpha) = \frac{1}{Z(\alpha)} \prod_{i=1}^{K} \theta_i^{(\alpha_i - 1)} \tag{84}$$

with parameters $\alpha_i > 0, i = 1, ..., K$. Note that the α_i do not need to be integers: $\alpha_i \in \mathbb{R}$.

The function $Z(\alpha)$ is a normalization factor similar to the $M(n)$ factor used in the Multinomial distribution, but now extended to real numbers:

$$Z(\alpha) = \frac{\prod_{i=1}^{K} \Gamma(\alpha_i)}{\Gamma(A)} \tag{85}$$

$$A = \alpha_1 + \alpha_2 + ... + \alpha_K \tag{86}$$

The function $\Gamma()$ is a generalization of the factorial function for real numbers:

$$\Gamma(x + 1) = x\, \Gamma(x) \tag{87}$$

For n integer:

$$\Gamma(n + 1) = n\, \Gamma(n) = n\, (n - 1)\, \Gamma(n - 1) = n(n - 1)(n - 2)...1 = n! \tag{88}$$

2.9.2 Bayesian Inference of the Posterior Distribution

We start with the Bayesian Inference equation (31), where the model is assumed as given (so we don't have to write it down explicitly), and my data D is the vector of counts $n = \vec{n} = n_1, ...n_K$ summing to N:

$$P(\theta|n) = \frac{P(n|\theta)\ P(\theta)}{P(n)} \tag{89}$$

Since the likelihood $P(n|\theta)$ follows a multinomial distribution (from section 2.8), and the prior $P(\theta)$ follows a Dirichlet distribution (sec. 2.9.1):

$$\begin{aligned} P(\theta|n) &= \frac{\mathcal{M}(n;\theta)\ \mathcal{D}(\theta;\alpha)}{P(n)} && (90)\\ &= \frac{1}{P(n)}\ \frac{1}{M(n)} \prod_{i=1}^{K} \theta_i^{n_i}\ \frac{1}{Z(\alpha)} \prod_{i=1}^{K} \theta_i^{(\alpha_i-1)} && (91)\\ &= \frac{1}{P(n)M(n)Z(\alpha)}\ \prod_{i=1}^{K} \theta_i^{n_i}\ \theta_i^{(\alpha_i-1)} && (92)\\ &= \frac{1}{P(n)M(n)Z(\alpha)}\ \prod_{i=1}^{K} \theta_i^{(n_i+\alpha_i)-1} && (93) \end{aligned}$$

The product term looks like a Dirichlet distribution with parameters $\alpha_i' = n_i + \alpha_i$, except that the term $Z(\alpha') = Z(n+\alpha)$ is missing. So we include it in both numerator and denominator, so that the result is not altered:

$$\begin{aligned} P(\theta|n) &= \frac{Z(n+\alpha)}{P(n)M(n)Z(\alpha)Z(n+\alpha)}\ \prod_{i=1}^{K} \theta_i^{(n_i+\alpha_i)-1} && (94)\\ &= \frac{Z(n+\alpha)}{P(n)M(n)Z(\alpha)}\ \mathcal{D}(\theta; n+\alpha) && (95) \end{aligned}$$

Since the posterior distribution is a probability distribution, it must sum to one, therefore:

$$\begin{aligned} \int_\theta P(\theta|n)\ d\theta &= 1 \Rightarrow && (96)\\ \int_\theta \frac{Z(n+\alpha)}{P(n)M(n)Z(\alpha)}\ \mathcal{D}(\theta; n+\alpha)\ d\theta &= 1 \Rightarrow && (97)\\ \frac{Z(n+\alpha)}{P(n)M(n)Z(\alpha)}\ \int_\theta \mathcal{D}(\theta; n+\alpha)\ d\theta &= 1 && (98) \end{aligned}$$

We know that $\mathcal{D}(\theta; n+\alpha)$ is a distribution hence it must sum to one, thus

$$\int_\theta \mathcal{D}(\theta; n+\alpha)\ d\theta = 1 \tag{99}$$

Therefore:

$$\begin{aligned} \frac{Z(n+\alpha)}{P(n)M(n)Z(\alpha)} &= 1 \Rightarrow && (100)\\ Z(n+\alpha) &= P(n)M(n)Z(\alpha) \Rightarrow && (101)\\ P(n) &= \frac{Z(n+\alpha)}{Z(\alpha)M(n)} && (102) \end{aligned}$$

So when the evidence $P(n)$ obeys eq. (102) above, the posterior follows a Dirichlet distribution. Indeed by substituting eq. (102) in eq. (95) we immediately get:

$$P(\theta|n) \sim \mathcal{D}(\theta; n+\alpha) \tag{103}$$

which indeed states that the posterior follows a Dirichlet distribution with parameters $n + \alpha$.

2.9.3 Estimating the Average of the Posterior: Pseudocounts

Now that we know that the posterior $P(\theta|n)$ follows a Dirichlet distribution, we are insterested in the average of the posterior, which is the Posterior Mean Estimate (PME, sec. 2.6):

$$\theta^{\text{PME}} = \int_\theta \theta \; P(\theta|n) \; d\theta = \int_\theta \theta \; \mathcal{D}(\theta; n+\alpha) \; d\theta \tag{104}$$

Since θ is a vector, we will compute the average for each θ_i individually:

$$\begin{aligned}
\theta_i^{\text{PME}} &= \int_\theta \theta_i \; \mathcal{D}(\theta; n+\alpha) \; d\theta && (105)\\
&= \int_\theta \theta_i \; \frac{1}{Z(n+\alpha)} \; \prod_{j=1}^{K} \theta_j^{(n_j+\alpha_j)-1} \; d\theta && (106)\\
&= \frac{1}{Z(n+\alpha)} \; \int_\theta \theta_i \; \prod_{j=1}^{K} \theta_j^{(n_j+\alpha_j)-1} \; d\theta && (107)
\end{aligned}$$

It would be nice if we could introduce the term θ_i into the product $\prod_j$ somehow. This can be done by adding a binary variable $\delta_{ij} = 1$ when $i = j$, and $\delta_{ij} = 0$ otherwise:

$$\theta_i^{\text{PME}} = \frac{1}{Z(n+\alpha)} \; \int_\theta \prod_{j=1}^{K} \theta_j^{(n_j+\alpha_j+\delta_{ij})-1} \; d\theta \tag{108}$$

We now apply the same trick as we did for the Dirichlet prior: the product above looks almost like a Dirichlet, so let's force it into one by multiplying the missing term up and down:

$$\begin{aligned}
\theta_i^{\text{PME}} &= \frac{1}{Z(n+\alpha)} \; \int_\theta \frac{Z(n+\alpha+\delta_i)}{Z(n+\alpha+\delta_i)} \; \prod_{j=1}^{K} \theta_j^{(n_j+\alpha_j+\delta_{ij})-1} \; d\theta && (109)\\
&= \frac{1}{Z(n+\alpha)} \; \int_\theta Z(n+\alpha+\delta_i) \; \mathcal{D}(\theta; n+\alpha+\delta_i) \; d\theta && (110)\\
&= \frac{Z(n+\alpha+\delta_i)}{Z(n+\alpha)} \; \int_\theta \mathcal{D}(\theta; n+\alpha+\delta_i) \; d\theta && (111)
\end{aligned}$$

We end up again with the integral of a distribution, which is one as we have seen before. Thus:

$$\theta_i^{\text{PME}} = \frac{Z(n+\alpha+\delta_i)}{Z(n+\alpha)} \tag{112}$$

Developing each term:

$$\begin{aligned}
Z(n+\alpha+\delta_i) &= \frac{\prod_{j=1}^{K} \Gamma(n_j+\alpha_j+\delta_{ij})}{\Gamma(N+A+1)} && (113)\\
&= \frac{\Gamma(n_1+\alpha_1)\Gamma(n_2+\alpha_2)...\Gamma(n_i+\alpha_i+1)...\Gamma(n_k+\alpha_k)}{\Gamma(N+A+1)} && (114)\\
Z(n+\alpha) &= \frac{\prod_{j=1}^{K} \Gamma(n_j+\alpha_j)}{\Gamma(N+A)} && (115)\\
&= \frac{\Gamma(n_1+\alpha_1)\Gamma(n_2+\alpha_2)...\Gamma(n_i+\alpha_i)...\Gamma(n_k+\alpha_k)}{\Gamma(N+A)} && (116)
\end{aligned}$$

There are many common terms in both equations, so when dividing one by the other most of the terms cancel out, leading to:

$$\begin{aligned}
\theta_i^{\text{PME}} &= \frac{Z(n+\alpha+\delta_i)}{Z(n+\alpha)} = \frac{\Gamma(n_i+\alpha_i+1)\;\Gamma(N+A)}{\Gamma(n_i+\alpha_i)\;\Gamma(N+A+1)} && (117)\\
&= \frac{(n_i+\alpha_i)\Gamma(n_i+\alpha_i)\;\Gamma(N+A)}{\Gamma(n_i+\alpha_i)\;(N+A)\Gamma(N+A))} \Rightarrow && (118)
\end{aligned}$$

$$\boxed{\theta_i^{\text{PME}} = \frac{n_i + \alpha_i}{N + A}} \tag{119}$$

which tells us that the average of the posterior distribution (or posterior mean estimate) is computed using **pseudocounts** α_i. The posterior mean estimate fixes the problem with the maximum likelihood estimator for θ (eq. (83)), in which observations with zero count ($n_i = 0$) cancel out the entire probability estimation, which is not reasonable.

2.10 Dealing with Heterogeneity: The Dirichlet Mixture

In the previous section we have performed Bayesian inference for a case where there was no grouping in the data: the estimated frequences of amino acids were valid for any position in the alignment. In that case one probability distribution was good enough to describe the entire data and parameter space.

Consider now a multiple sequence alignment involving G-Protein coupled receptors (GPCRs). These proteins alternate hydrophylic and hydrophobic domains: the priors for these domains may be different due to the different frequencies of amino acids in each domain (hydrophobic amino acids are more common in the hydrophobic part of the protein, and hydrophilic AAs in the hydrophilic domain). So we would need two different priors for each type of domain:

$$\begin{aligned} &\text{Prior for hydrophobic domains:} \quad P(\theta^+) = \mathcal{D}(\theta^+; \alpha^+) \\ &\text{Prior for hydrophilic domains:} \quad P(\theta^-) = \mathcal{D}(\theta^-; \alpha^-) \end{aligned}$$

However in practice we don't know when a domain starts or stops in the alignment, so we don't have the information to compute two separate priors. The solution is to generate a single vector θ from a **Dirichlet Mixture** of m different sources S ($m = 2$ in the GPCR example): a weighted average of m Dirichlet distributions with different parameters α^k, one per source S:

$$P(\theta) = \sum_k P(S = k)\ \mathcal{D}(\theta; \alpha^k) = \sum_k q_k\ \mathcal{D}(\theta; \alpha^k) \tag{120}$$

where $q_k = P(S = k)$ is the frequency of source $S = k$.

For the GPCR example this would become:

$$P(\theta) = q^+\ \mathcal{D}(\theta; \alpha^+) + q^-\ \mathcal{D}(\theta; \alpha^-) \tag{121}$$

Now that we have the prior, we can compute the posterior distribution and its mean. Instead of using the Bayesian inference rule (eq. 31)) directly however, the derivations take a completely different path!! So before we begin, let's lay out some ingredients that we'll need later.

If we know the source $S = k$ then:

- The frequency of source $S = k$ is q_k as said before:

$$q_k = P(S = k) \tag{122}$$

- The **prior** ???? that we use, given that we know the column of the alignment (data vector n) and the source $S = k$, is the **posterior** that we computed in eq. (103), for the particular α_k:

$$P(\theta | S = k, n) = \mathcal{D}(\theta; n + \alpha^k) \tag{123}$$

- The **evidence**, given that I'm using a particular source $S = k$ is (just use α^k instead of α in eq. 102):

$$P(n | S = k) = \frac{Z(n + \alpha^k)}{Z(\alpha^k) M(n)} \tag{124}$$

Using marginalization:

$$P(\theta|n) = \sum_k P(\theta, S=k|n) = \sum_k P(\theta|S=k,n)\ P(S=k|n) \tag{125}$$

We know $P(\theta|S=k,n)$ from eq. (123). We can compute $P(S=k|n)$ using Bayes' rule:

$$P(S=k|n) = \frac{P(n|S=k)\ P(S=k)}{P(n)} \tag{126}$$

We know that $P(S=k) = q_k$ (eq. (122)), and we know $P(n|S=k)$ from eq. (124). $P(n)$ can be computed by marginalization and with the help of eq. (124):

$$\begin{aligned} P(n) &= \sum_l P(n|S=l)\ P(S=l) && (127)\\ &= \sum_l q_l\ \frac{Z(n+\alpha^l)}{Z(\alpha^l)\ M(n)} && (128)\\ &= \frac{1}{M(n)} \sum_l q_l\ \frac{Z(n+\alpha^l)}{Z(\alpha^l)} && (129)\\ &\qquad \text{for} \quad l = 1, ..., m \end{aligned}$$

Substitute eqs. (122), (124), and (129) in eq. (126):

$$\begin{aligned} P(S=k|n) &= \frac{q_k\ Z(n+\alpha^k)}{\left(\frac{1}{M(n)} \sum_l q_l\ \frac{Z(n+\alpha^l)}{Z(\alpha^l)} \right) Z(\alpha^k)\ M(n)} \quad \Rightarrow && (130)\\ P(S=k|n) &= \frac{q_k\ Z(n+\alpha^k)/Z(\alpha^k)}{\sum_l q_l\ Z(n+\alpha^l)/Z(\alpha^l)} && (131) \end{aligned}$$

Now coming back to eq. (125):

$$P(\theta|n) = \sum_k \mathcal{D}(\theta; n+\alpha^k)\ P(S=k|n) \tag{132}$$

We know how to compute the term $P(S=k|n)$ using eq. (131), but we won't substitute it here since it is a bit bulky.

Using eq. (132), compute the posterior mean estimate for a given θ_i:

$$\begin{aligned} \theta_i^{\text{PME}} &= \int_\theta \theta_i\ P(\theta|n)\ d\theta && (133)\\ &= \int_\theta \theta_i \sum_k \mathcal{D}(\theta; n+\alpha^k)\ P(S=k|n)\ d\theta && (134)\\ &= \int_\theta \sum_k \theta_i\ \mathcal{D}(\theta; n+\alpha^k)\ P(S=k|n)\ d\theta && (135)\\ &= \sum_k \int_\theta \theta_i\ \mathcal{D}(\theta; n+\alpha^k)\ P(S=k|n)\ d\theta && (136)\\ &= \sum_k P(S=k|n) \int_\theta \theta_i\ \mathcal{D}(\theta; n+\alpha^k)\ d\theta && (137) \end{aligned}$$

The manipulations above were possible because the integral of a sum is the sum of integrals, θ_i does not depend on k, and $P(S=k|n)$ does not depend on θ.

The remaining integral in eq. (137) is the same as eq. (105) for $\alpha = \alpha_k$, and we already know how to solve it: θ_i^{PME} can be calculated using the pseudocounts as demonstrated in eq. (119). Therefore, for a given source $S = k$:

$$\theta_i^{\text{PME,k}} = \int_\theta \theta_i \, \mathcal{D}(\theta; n + \alpha^k) \, d\theta = \frac{n_i + \alpha_i^k}{N + A^k} \tag{138}$$

Finally, substitute eq. (138) in eq. (137) to get:

$$\boxed{\theta_i^{\text{PME}} = \sum_k P(S = k|n) \, \frac{n_i + \alpha_i^k}{N + A^k}} \tag{139}$$

which means that the posterior mean estimate of parameters in a Dirichlet mixture is the weighted sum of the PME estimation using pseudocounts assuming a specific source k, weighted by the posterior probability that we chose the right source k given the data n from the alignment.

3 Part 3: Hidden Markov Models (HMM)

Hidden Markov Models (HMMs) are used to detect patterns in sequences in a probabilistic way. Another way to detect patterns could be via regular expressions, but these don't capture exceptions, consensus, or higher probabilities for certain amino acids or certain sequence positions than others, etc.

3.1 Log Odds to Score Sequences

In section 1.1 we have seen that a log-odds ratio is commonly used to compute scores in amino acid and nucleotide substitution matrices. A similar log odds ratio is used to score entire sequences as well. The **log odds** for a nucleotide sequence S of length L is:

$$\log \frac{P(S)}{0.25^L} = \log(P(S)) - L \ \log(0.25) \tag{140}$$

This metric is used to scale and normalize by a random model. Scaling is important in order not to penalize longer sequences (that would otherwise get lower probability, because they get more numbers smaller than one multiplied together). Normalizing by a random model is important to highlight the real patterns from the background noise.

3.2 Markov Chain

Figure 8 shows an example of Markov Chain (MC). It is a state machine with one symbol per state. After emitting the symbol, the machine transitions to one of the possible next states with a given probability. A sequence $x = x_1, ...x_L$ can be generated from the MC state machine in this way. Each symbol x_i belongs to the alphabet $\mathcal{A}$ that also defines the set of states of the state machine: $x_i \in \mathcal{A}$, e.g. $\mathcal{A} = \{A, C, T, G\}$ for a nucleotide sequence x.

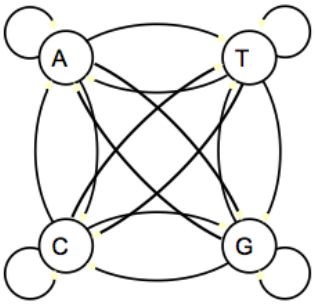

Figure 8: Example of Markov Chain

The transition probability a_{st} to go from state s to state t is:

$$\boxed{a_{st} = P(x_i = t | x_{i-1} = s)} \tag{141}$$

3.2.1 Markov Property

The key assumption of a Markov Chain is that the probability of each symbol x_i depends only on the previous symbol x_{i-1}. This is called the Markov property:

$$\boxed{P(x_i | x_{i-1}, ..., x_1) = P(x_i | x_{i-1}) = a_{x_{i-1}, x_i}} \tag{142}$$

Therefore the probability of the whole sequence x becomes:

$$\begin{aligned} P(x) &= P(x_L, x_{L-1}, ..., x_1) \\ &= P(x_L|x_{L-1}, ..., x_1)P(x_{L-1}, ..., x_1) \\ &= P(x_L|x_{L-1}, ..., x_1)P(x_{L-1}|x_{L-2}..., x_1)P(x_{L-2}..., x_1) \\ &= ... \\ &= P(x_L|x_{L-1})P(x_{L-1}|x_{L-2})...P(x_2|x_1)P(x_1) \Rightarrow \\ P(x) &= P(x_1)\prod_{i=2}^{L} P(x_i|x_{i-1}) \end{aligned} \tag{143}$$

Hence:

$$\boxed{P(x) = P(x_1)\prod_{i=2}^{L} a_{x_{i-1},x_i}} \tag{144}$$

We can simplify this formula by adding an initial symbol $x_0 = \alpha$ with $P(x_0) = 1$:

$$\boxed{P(x) = \prod_{i=1}^{L} a_{x_{i-1},x_i}} \tag{145}$$

3.3 Hidden Markov Model (HMM)

In a Hidden Markov Model, each state can emit a number of symbols, each symbol is emitted with a probability, that can differ from one state to another. Figure 9 shows an example.

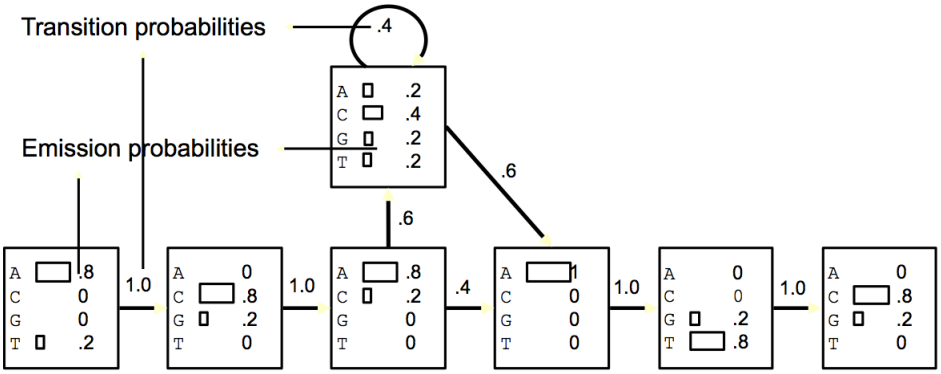

Figure 9: Example of Hidden Markov Model

Each element x_i of a sequence $x = x_1, ..., x_L$ is produced by state π_i. The path through the state machine is the sequence of states $\pi = \pi_1, ..., \pi_L$ that produced x. Now given the sequence x we can no longer identify the precise state from where each symbol x_i was emitted, since several states could have produced it. This is why this MC is "hidden".

The HMM is characterized by *state transition* probabilities a_{kl} and *symbol emission* probabilities e_k:

$$a_{kl} = P(\pi_i = l \mid \pi_{i-1} = k) \tag{146}$$

$$e_k(b) = P(x_i = b \mid \pi_i = k) \tag{147}$$

- The transition probability a_{kl} is the probability that the current state π_i is l given that the previous state π_{i-1} was k.
- The emission probability $e_k(b)$ is the probability that symbol b is emitted when we are in state $\pi_i = k$ (that is, given that the current state is k).

The HMM starts in state $\pi_0 = \alpha$ and ends in state $\pi_{L+1} = \omega$.

3.3.1 Markov Property in HMM

It is useful to rewrite the Markov property expressed by eq. 142 in the HMM context:

$$P(\pi_i \mid \pi_{i-1}, ..., \pi_1) = P(\pi_i \mid \pi_{i-1}) = a_{\pi_{i-1},\pi_i} \tag{148}$$

$$P(\pi_i \mid x_{i-1}, ..., x_1, \pi_{i-1}, ..., \pi_1) = P(\pi_i \mid \pi_{i-1}) = a_{\pi_{i-1},\pi_i} \tag{149}$$

$$P(x_i \mid x_{i-1}, ..., x_1, \pi_i, \pi_{i-1}, ..., \pi_1) = P(x_i \mid \pi_i) = e_{\pi_i}(x_i) \tag{150}$$

The probability of a path $\pi = \pi_0, \pi_1, ..., \pi_L, \pi_{L+1}$ can be calculated using the product rule and the Markov property from eq. (148):

$$\begin{aligned} P(\pi) &= P(\pi_{L+1}, \pi_L, \pi_{L-1}, ..., \pi_1, \pi_0) \\ &= P(\pi_{L+1}|\pi_L, ..., \pi_0)P(\pi_L, ..., \pi_0) \\ &= ... \\ &= P(\pi_{L+1}|\pi_L)P(\pi_L|\pi_{L-1})...P(\pi_1|\pi_0)P(\pi_0) \Rightarrow \end{aligned}$$

$$\boxed{P(\pi) = \prod_{i=0}^{L} P(\pi_{i+1}|\pi_i) = \prod_{i=0}^{L} a_{\pi_i,\pi_{i+1}}} \tag{151}$$

for $P(\pi_0) = P(\pi_0 = \alpha) = 1$.

Therefore the probability of a path $\pi = \pi_0, \pi_1, ..., \pi_L, \pi_{L+1}$ is simply the product of the transition probabilities $a_{\pi_i,\pi_{i+1}}$ along the path.

The joint probability of a sequence x and a path π is:

$$P(x, \pi) = P(x|\pi)P(\pi) \tag{152}$$

$P(\pi)$ is given by eq. (151), so let's calculate $P(x|\pi)$, using the product rule again, and the Markov assumption of eq. (150):

$$\begin{aligned} P(x|\pi) &= P(x_L, x_{L-1}, ..., x_1 \mid \pi_{L+1}, \pi_L, \pi_{L-1}, ..., \pi_1, \pi_0) \\ &= P(x_L \mid x_{L-1}, ..., x_1, \pi_{L+1}, ..., \pi_0)P(x_{L-1}, ..., x_1, \pi_{L+1}, ..., \pi_0) \\ &= P(x_L \mid \pi_L,)P(x_{L-1} \mid x_{L-2}, ..., x_1, \pi_{L+1}, ..., \pi_0)P(x_{L-2}, ..., x_1, \pi_{L+1}, ..., \pi_0) \\ &= P(x_L \mid \pi_L,)P(x_{L-1} \mid \pi_{L-1})...P(x_1|\pi_1)P(x_0|\pi_0) \Rightarrow \end{aligned}$$

$$\boxed{P(x|\pi) = \prod_{i=0}^{L} P(x_i|\pi_i) = \prod_{i=0}^{L} e_{\pi_i}(x_i)} \tag{153}$$

Hence the probability of a sequence x given a path π is simply the product of the emission probabilities of the symbols x_i of x along the path π.

The joint probability $P(x, \pi)$ can now be obtained by substituting eqs. (151) and (153) in (152):

$$P(x, \pi) = P(x|\pi)P(\pi) = \prod_{i=0}^{L} e_{\pi_i}(x_i) \prod_{i=0}^{L} a_{\pi_i,\pi_{i+1}} \Rightarrow \tag{154}$$

$$\boxed{P(x, \pi) = \prod_{i=0}^{L} e_{\pi_i}(x_i)\ a_{\pi_i,\pi_{i+1}}} \tag{155}$$

which simply states that the joint probability of the sequence x and the path π is the product of the emission and transition probabilities along the path.

3.4 Viterbi Algorithm

Find the most probable path π^* for a given sequence x:

$$\boxed{\pi^* = \underset{\pi}{\operatorname{argmax}}\, P(x, \pi)} \tag{156}$$

Equation (155) gives a way to calculate $P(x, \pi)$, however we don't have a path but would like to find the most probable one, without computing all possible paths, so eq. (155) cannot be applied directly to compute eq. (156). A dynamic programming algorithm is used instead.

If we apply the product rule:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, P(x, \pi) = \underset{\pi}{\operatorname{argmax}}\, P(\pi|x)P(x) = \underset{\pi}{\operatorname{argmax}}\, P(\pi|x) \tag{157}$$

we see that the term $P(x)$ does not depend on π hence does not influence the maximum and can be eliminated. However starting with eq. (157) instead of eq. (156) doesn't work to derive the Viterbi algorithm, because we get terms that we don't know how to compute. So we stick to eq. (156).

Construct a partial solution up to position $\pi_i = k$:

$$v_k(i) = \max_{\pi_1,...\pi_{i-1}} P(x_1, ..., x_i, \pi_1, ..., \pi_{i-1}, \pi_i = k) \tag{158}$$

for all possible k.

Use recursion to build the next step $v_l(i+1)$ of the solution as a function of the current step $v_k(i)$:

$$v_l(i+1) = \max_{\pi_1,...\pi_i=k} P(x_1, ..., x_{i+1}, \pi_1, ...\pi_i = k, \pi_{i+1} = l) \tag{159}$$

where k is the optimum state chosen in the maximization step $v_k(i)$, and l is the new optimum to be determined for the next step $v_l(i+1)$.

Use the product rule and the Markov property to break down $v_l(i+1)$ into known elements that can be computed:

$$\begin{aligned}
v_l(i+1) &= \max_{\pi_1,...\pi_i=k} P(x_1, ..., x_{i+1}, \pi_1, ...\pi_i = k, \pi_{i+1} = l) \\
&= \max_{\pi_1,...\pi_i=k} P(x_{i+1}|x_1, ..., x_i, \pi_1, ...\pi_i = k, \pi_{i+1} = l)P(x_1, ..., x_i, \pi_1, ...\pi_i = k, \pi_{i+1} = l) \\
&= \max_{\pi_1,...\pi_i=k} P(x_{i+1}|\pi_{i+1} = l)P(\pi_{i+1} = l|x_1, ..., x_i, \pi_1, ...\pi_i = k)P(x_1, ..., x_i, \pi_1, ...\pi_i = k) \\
&= \max_{\pi_1,...\pi_i=k} P(x_{i+1}|\pi_{i+1} = l)P(\pi_{i+1} = l|\pi_i = k)P(x_1, ..., x_i, \pi_1, ...\pi_i = k) \\
&= \max_{\pi_1,...\pi_i=k} e_l(x_{i+1})\, a_{kl}\, P(x_1, ..., x_i, \pi_1, ...\pi_i = k)
\end{aligned} \tag{160}$$

Now break the maximization step in two parts: maximize over $\pi_1, ..., \pi_{i-1}$ and then maximize over $\pi_i = k$:

$$v_l(i+1) = \max_{\pi_i=k} \max_{\pi_1,...\pi_{i-1}} e_l(x_{i+1})\, a_{kl}\, P(x_1, ..., x_i, \pi_1, ...\pi_i = k) \tag{161}$$

The term $e_l(x_{i+1})$ does not depend on $\pi_1, ..., \pi_i$ so can be take out of both max functions. Moreover the term a_{kl} does not depend on $\pi_1, ...\pi_{i-1}$ hence can be taken out of the inner max:

$$v_l(i+1) = e_l(x_{i+1}) \max_{\pi_i=k} \left(a_{kl} \max_{\pi_1,...\pi_{i-1}} P(x_1, ..., x_i, \pi_1, ...\pi_i = k) \right) \tag{162}$$

Now we recognize the last max term of eq. (162) above as $v_k(i)$, arriving at the **Viterbi recursion** formula:

$$\boxed{v_l(i+1) = e_l(x_{i+1}) \max_{\pi_i=k} (\, a_{kl}\, v_k(i)\,)} \tag{163}$$

Viterbi finds the most probable path π^*, but since the number of possible paths is usually huge, the probability of π^* might actually be very small (even if bigger than the other even smaller probabilities for the other paths), so in practice the "optimal" path π^* is of little use.

3.5 Forward Algorithm

The forward algorithm computes the probability of a sequence x of length L, $x = x_1, ...x_L$, for all possible paths π.

$$\boxed{P(x) = \sum_{\pi} P(x, \pi)} \tag{164}$$

As in the Viterbi case, we don't want to use eq. (155) directly since it would require enumerating all possible paths. Dynamic programming is again the solution here.

Start with a partial solution $f_k(i)$, the probability of subsequence $x_1, ..., x_i$ when x_i is emitted by state k:

$$f_k(i) = P(x_1, ..., x_i, \pi_i = k) \tag{165}$$

for all possible combinations of $\pi_1, ...\pi_{i-1}$ (these $\pi_1, ...\pi_{i-1}$ are not given, hence they do not appear in eq. (165)).

Define the next step $f_l(i+1)$:

$$f_l(i+1) = P(x_1, ..., x_{i+1}, \pi_{i+1} = l) \tag{166}$$

We do not know $\pi_i = k$, so we must use marginalization to sum over all possible values of k:

$$f_l(i+1) = \sum_{k=1}^{K} P(x_1, ..., x_i, x_{i+1}, \pi_i = k, \pi_{i+1} = l) \tag{167}$$

where K is the number of states in the HMM.

Apply the product rule to eq. (167) with the goal to isolate $f_k(i)$, so we pick x_{i+1} first:

$$f_l(i+1) = \sum_{k=1}^{K} P(x_{i+1}|x_1, ..., x_i, \pi_i = k, \pi_{i+1} = l) P(x_1, ..., x_i, \pi_i = k, \pi_{i+1} = l) \tag{168}$$

Now use the Markov property:

$$f_l(i+1) = \sum_{k=1}^{K} P(x_{i+1}|\pi_{i+1} = l) P(x_1, ..., x_i, \pi_i = k, \pi_{i+1} = l) \tag{169}$$

Apply the product rule again, now picking $\pi_{i+1} = l$ to make $f_k(i)$ appear:

$$f_l(i+1) = \sum_{k=1}^{K} P(x_{i+1}|\pi_{i+1} = l) P(\pi_{i+1} = l|x_1, ..., x_i, \pi_i = k) P(x_1, ..., x_i, \pi_i = k) \tag{170}$$

Use the Markov property:

$$f_l(i+1) = \sum_{k=1}^{K} P(x_{i+1} \mid \pi_{i+1} = l)\ P(\pi_{i+1} = l \mid \pi_i = k)\ P(x_1, ..., x_i, \pi_i = k) \tag{171}$$

We now recognize the terms of eq. (171) as $e_l(x_{i+1})$, a_{kl}, and $f_k(i)$, respectively:

$$f_l(i+1) = \sum_{k=1}^{K} e_l(x_{i+1})\ a_{kl}\ f_k(i) \tag{172}$$

The term $e_l(x_{i+1})$ does not depend on k, hence can be taken out of the sum, leading to the final recursion equation for the forward algorithm:

$$\boxed{f_l(i+1) = e_l(x_{i+1}) \sum_{k=1}^{K} a_{kl}\ f_k(i)} \tag{173}$$

3.6 Backward Algorithm

This algorithm computes the probability that state $\pi_i = k$ was the one that generated symbol x_i, given the entire sequence $x = x_1, ...x_L$:

$$\boxed{P(\pi_i = k|x) = P(\pi_i = k \mid x_1, ...x_L)} \tag{174}$$

Note that $P(\pi_i = k|x) \neq P(x_i|\pi_i = k)$:

- $P(\pi_i = k|x)$ is the probability that state k, out of all possible states of the HMM, was the one responsible for generating x_i in x (but we don't know for sure which state actually generated x_i);
- $P(x_i|\pi_i = k) = e_k(x_i)$ is the emission probability of x_i from state k once we *know* that we are in state k (state k is *given*).

Using the product rule:

$$P(x, \pi_i = k) = P(\pi_i = k|x)P(x) \Rightarrow \tag{175}$$

$$P(\pi_i = k|x) = \frac{P(x, \pi_i = k)}{P(x)} \tag{176}$$

The denominator $P(x)$ can be computed by the foward algorithm. Let's then compute the numerator $P(x, \pi_i = k)$ using the product rule and the Markov property:

$$\begin{aligned} P(x, \pi_i = k) &= P(x_1, ..., x_i, x_{i+1}, ..., x_L, \pi_i = k) \\ &= P(x_1, ..., x_i, \pi_i = k, x_{i+1}, ..., x_L) \\ &= P(x_{i+1}, ..., x_L|x_1, ..., x_i, \pi_i = k)P(x_1, ..., x_i, \pi_i = k) \\ &= P(x_{i+1}, ..., x_L|\pi_i = k)P(x_1, ..., x_i, \pi_i = k) \end{aligned} \tag{177}$$

We recognize the last term of eq. (177) as $f_k(i) = P(x_1, ..., x_i, \pi_i = k)$. Call the remaining term $b_k(i)$:

$$b_k(i) = P(x_{i+1}, ..., x_L|\pi_i = k) \tag{178}$$

Equation (177) then becomes:

$$P(x, \pi_i = k) = f_k(i)b_i(i) \tag{179}$$

Note that we cannot eliminate $\pi_i = k$ from eqs. (177) or (178) using the Markov property: even though $x_{i+1}, ..., x_L$ do not depend directly on $\pi_i = k$, the term π_{i+i} upon which x_{i+1} depends is unknown, and π_{i+i} on its turn depends on π_i, hence we must leave $\pi_i = k$ there. If we knew π_{i+i}, then we could have scratched out $\pi_i = k$.

We must now build a recursion from backwards to compute $b_k(i)$:

$$b_l(i + 1) = P(x_{i+2}, ..., x_L|\pi_{i+1} = l) \tag{180}$$

We don't know $\pi_{i+1} = l$, hence we must use marginalization to compute all possibilities for it:

$$b_k(i) = \sum_{l=1}^{K} P(x_{i+1}, x_{i+2}, ..., x_L, \pi_{i+1} = l|\pi_i = k) \tag{181}$$

Since $\pi_{i+1} = l$ is unknown, hence not given, it appears at the front part of the probability rule, not at the tail (the "given" part).

Now apply the product rule $P(A, B|D) = P(A|B, D)P(B|D)$ with:
$A \leftarrow (x_{i+1}, x_{i+2}, ..., x_L)$, $B \leftarrow (\pi_{i+1} = l)$, $D \leftarrow (\pi_i = k)$:

$$b_k(i) = \sum_{l=1}^{K} P(x_{i+1}, x_{i+2}, ..., x_L \mid \pi_{i+1} = l, \pi_i = k)\ P(\pi_{i+1} = l \mid \pi_i = k) \tag{182}$$

We recognize the last term of the above equation as a_{kl}. Moreover now we can eliminate $\pi_i = k$ from the first term using the Markov property, since $\pi_{i+1} = l$ is given there:

$$b_k(i) = \sum_{l=1}^{K} P(x_{i+1}, x_{i+2}, ..., x_L \mid \pi_{i+1} = l)\ a_{kl} \tag{183}$$

Now apply the product rule $P(A, B|D) = P(A|B, D)P(B|D)$ again with:
$A \leftarrow (x_{i+1})$, $B \leftarrow (x_{i+2}, ..., x_L)$, $D \leftarrow (\pi_{i+1} = l)$:

$$\begin{aligned} b_k(i) &= \sum_{l=1}^{K} P(x_{i+1} \mid x_{i+2}, ..., x_L, \pi_{i+1} = l)\ P(x_{i+2}, ..., x_L \mid \pi_{i+1} = l)\ a_{kl} \\ &= \sum_{l=1}^{K} P(x_{i+1} \mid \pi_{i+1} = l)\ b_l(i+1)\ a_{kl} \Rightarrow \end{aligned}$$

$$\boxed{b_k(i) = \sum_{l=1}^{K} e_l(x_{i+1})\ b_l(i+1)\ a_{kl}} \tag{184}$$

which is the final recursion formula for $b_k(i)$.

Now that we know how to compute $b_k(i)$, we can go back to eqs. (176) and (179) to tackle our original goal to compute $P(\pi_i = k|x)$:

$$\boxed{P(\pi_i = k|x) = \frac{P(x, \pi_i = k)}{P(x)} = \frac{f_k(i)\ b_k(i)}{P(x)}} \tag{185}$$

Now all the terms $f_k(i)$, $b_k(i)$, and $P(x)$ can be computed.

3.7 Posterior decoding sl 20

Instead of using the most probable path (Viterbi), use the path of the most probable states:

$$\hat{\pi}_i = \underset{k}{\text{argmax}}\, P(\pi_i = k|x) \tag{186}$$

where $P(\pi_i = k|x)$ can be computed by the backward algorithm. Caution however because the path $\hat{\pi}_i$ can be illegal!!

3.8 Parameter estimation with known paths sl 25

Given a training set $\mathcal{D}$ of N sequences $x^1, ...x^N$: $\mathcal{D} = \{x^1, ...x^N\}$, we would like to find the set of parameters θ that maximize the likelihood of seeing these sequences: $\text{argmax}_\theta\, P(x|\theta)$.

Score the model using the log likelihood of the parameters θ given the training data:

$$\text{Score}(\mathcal{D}, \theta) = \log P(x^1, ..., x^N|\theta) = \sum_{j=1}^{N} \log P(x^j|\theta) \tag{187}$$

The parameters in this case are the transition and emission probabilities $\theta = \{a_{kl}, e_k(b)\}$ for all states and symbols. With known paths, it suffices to walk along these paths for all training sequences, and to count how often a transition to state l is used from state k (A_{kl}), and how often a symbol b is produced from state k ($E_k(b)$. The probabilities a_{kl} can then be obtained by ML estimation as we saw in section 2.8.1

$$a_{kl} = \frac{A_{kl}}{\sum_{l'} A_{kl'}} \tag{188}$$

$$e_k(b) = \frac{E_k(b)}{\sum_{b'} E_k(b')} \tag{189}$$

or better, by using pseudocounts as explained in section 2.9.3: in this case, A_{kl} and $E_k(b)$ can include the pseudocounts.

3.9 Parameter estimation with unknown paths sl 27

When the paths π are unknown, we cannot simply count the frequencies as we did in the previous section, because we don't know where to go in the HMM, given just the sequences x. The solution here is to estimate the parameters and the paths using an iterative method, via progressive refinement from a initially coarse solution. Two algorithms can take care of this: Viterbi training and Baum-Welch training.

3.9.1 Viterbi training

Viterbi training makes use of the Viterbi algorithm to compute the most probable path π^* for each traning sequence, given a coarse initial assignment of $\theta = \{a_{kl}, e_k(b)\}$ (for instance, initially all states and all transitions are equally probable). These paths are then used to estimate new frequencies via counting, using eqs. (188) and (189). These new frequencies are then used to obtain new optimum paths, and so on. This process is repeated using the new frequencies and the optimum paths from the previous iteration to refine the frequencies and paths for the next iteration, until convergence, leading to a simultaneous estimation of the optimum path and the optimum parameters:

$$x, \theta^0 \rightarrow \text{Viterbi} \rightarrow x, \pi \rightarrow \text{count} \rightarrow x, \theta^1 \rightarrow \text{Viterbi} \rightarrow ... \rightarrow x, \pi^*, \theta^* \tag{190}$$

The algorithm is guaranteed to converge to a set of parameters such that the same path results from the same parameters, so at some point things stop changing, that is, the algorithm converges. However it is too greedy (takes the most probable path at each step, which might ignore a vast amount of good paths). Moreover it does not compute the maximum likelihood estimate for the parameters:

$$\theta^* \neq \underset{\theta}{\text{argmax}}\, P(x_1, ..., x_N|\theta) \tag{191}$$

where θ^* is the optimum θ found by Viterbi training, and $x_1, ..., x_N$ are the sequences from the training set.

3.9.2 Baum-Welch training sl 28

Instead of taking only the most probable path at each step and ignoring all the others as done by Viterbi training, the Baum-Welch algorithm estimates the parameters $\theta = \{a_{kl}, e_k(b)\}$ by calculating the *expected* number of times each transition or emission is used, for all possible paths. Of course, it must do this without enumerating all possible paths explicitly, which can be achieved by using the forward and the backward algorithms.

Remember that the forward algorithm computes the probability of a sequence for all possible paths, and the backward algorithm computes the probability that a particular state was the one that produced a particular symbol in a sequence.

Moreover in section 3.5 we saw that the forward algorithm is computed using the existing parameters $\theta = \{a_{kl}, e_k(b)\}$ that were then given, but that we are now going go estimate. So actually the forward algorithm computes $P(x|\theta)$ which is the likelihood of the data x given the parameters θ. The partial solution $f_k(i)$ then becomes:

$$f_k(i) = P(x_1, ..., x_i, \pi_i = k|\theta) \tag{192}$$

Similarly, the backward algorithm computes $P(\pi_i = k|x, \theta)$, and its partial solution is:

$$b_l(i+1) = P(x_{i+2}, ..., x_L|\pi_{i+1} = l, \theta) \tag{193}$$

The probability that transition $k \to l$ is used in a particular sequence x is given by:

$$P(\pi_i = k, \pi_{i+1} = l|x, \theta) = \frac{P(\pi_i = k, \pi_{i+1} = l, x|\theta)}{P(x|\theta)} \tag{194}$$

Initially we don't know any parameters, so we set $\theta = \theta^0$ to their pseudocount values. $P(x|\theta)$ can be computed by the forward algorithm so we focus on the other term:

$$\begin{aligned} P(\pi_i = k, \pi_{i+1} = l, x|\theta) &= P(x_1, ..., x_L, \pi_i = k, \pi_{i+1} = l|\theta) && (195)\\ &= P(x_1, ..., x_i, \pi_i = k, \pi_{i+1} = l, x_{i+1}, ..., x_L|\theta) && (196)\\ &= P(x_{i+1}, ..., x_L, \pi_{i+1} = l|x_1, ..., x_i, \pi_i = k, \theta) \\ &\quad P(x_1, ..., x_i, \pi_i = k|\theta) && (197)\end{aligned}$$

We recognize the last line as $f_k(i) = P(x_1, ..., x_i, \pi_i = k|\theta)$, so let's develop the other one:

$$\begin{aligned} &P(x_{i+1}, ..., x_L, \pi_{i+1} = l|x_1, ..., x_i, \pi_i = k, \theta) = \\ &P(x_{i+1}, ..., x_L, \pi_{i+1} = l|\pi_i = k, \theta) = \\ &P(x_{i+1}|x_{i+2}, ..., x_L, \pi_{i+1} = l, \pi_i = k, \theta)P(x_{i+2}, ..., x_L, \pi_{i+1} = l|\pi_i = k, \theta) = \\ &P(x_{i+1}|\pi_{i+1} = l, \theta)P(x_{i+2}, ..., x_L|\pi_{i+1} = l, \pi_i = k, \theta)P(\pi_{i+1} = l|\pi_i = k, \theta) = \\ &P(x_{i+1}|\pi_{i+1} = l, \theta)P(x_{i+2}, ..., x_L|\pi_{i+1} = l, \theta)P(\pi_{i+1} = l|\pi_i = k, \theta) = \\ &e_l(x_{i+1})b_l(i+1)a_{kl} \end{aligned} \tag{198}$$

Combining all together we get:

$$P(\pi_i = k, \pi_{i+1} = l|x, \theta) = \frac{a_{kl}\ e_l(x_{i+1})\ f_k(i)\ b_l(i+1)}{P(x|\theta)} \tag{199}$$

All these terms can be computed using the forward and backward algorithms, using the estmations of parameters from the previous iteration.

The expected number of times that transition a_{kl} is used can be estimated by summing eq. (199) for all positions in all training sequences:

$$\begin{aligned} A_{kl} &= \sum_{j=1}^{N}\sum_{i=1}^{L} P(\pi_i = k, \pi_{i+1} = l|x_i^j, \theta) && (200)\\ &= \sum_{j=1}^{N} \frac{1}{P(x^j|\theta)} \sum_{i=1}^{L} a_{kl}\ e_l(x_{i+1}^j)\ f_k^j(i)\ b_l^j(i+1) && (201)\end{aligned}$$

The expected emission count is obtained in a similar way, by summing over all positions i in all sequences x^j where the symbol x_i^j at position i in sequence j is b:

$$\begin{aligned} E_k(b) &= \sum_j \sum_{i|x_i^j=b} P(\pi_i = k|x^j, \theta) && (202)\\ &= \sum_j \frac{1}{P(x^j|\theta)} \sum_{i|x_i^j=b} f_k^j(i)\ b_k^j(i) && (203)\end{aligned}$$

Using these expected counts, new transition and emission parameters can be estimated for use in the next iteration, using eqs. (188) and (189), repeated here for convenience:

$$\begin{aligned} a_{kl} &= \frac{A_{kl}}{\sum_{l'} A_{kl'}} && (204)\\ e_k(b) &= \frac{E_k(b)}{\sum_{b'} E_k(b')} && (205)\end{aligned}$$

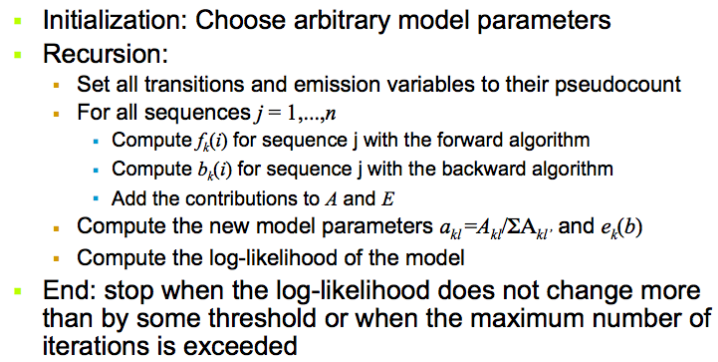

Figure 10: The Baum-Welch training algorithm

The resulting algorithm is shown in figure 10. At each iteration, the Baum-Welch algorithm improves the likelihood estimate of the parameters, in contrast with Viterbi training. We will see in section 4 that Baum-Welch training is a special case of Expectation Maximization (EM) algorithm.

A major limitation of Baum-Welch training is that it consumes a lot of data to estimate the parameters from scratch (when no prior knowledge is available): thousands of observations might be needed to produce a good model, and then thousands more to reduce the error by a factor 10. However, prior knowledge can greatly reduce the number of observations needed to get a sufficiently good estimation of parameters.

Another issue is the problem of numerical instability due to the product of small probabilities: simply taking log scale works well for Viterbi, but doesn't work well for the forward algorithm, because we get logs of sums, which are not so nice. The solution is to take approximations of algorithms, or define new variables that are in log-like scale.

3.10 HMM Applications

3.10.1 Profile HMMs

3.10.2 Gene Finding

predict the location of genes; often combined with homology search.

analysis of codon bias

GENSCAN: gene prediction using Hidden Semi-Markov Models

semi-markov chain: before moving to next state, can stay there for a certain period of time; number of discrete time steps drawn from a distribution: draw from that discrete distr to decide if you stay in that state or move on

Hidden Semi-Markov: enter state: each state may be a quite complex model: can put a whole HMM inside the state, and from this HMM you can generate a small subsequence before moving to the next state

4 Part 4: Expectation Maximization (EM)

In section 3.9.2 we have seen how the Baum-Welch algorithm progressively refines the estimation of model parameters θ, improving their likelihood estimate. In this section we will see how this notion can be generalized to an algorithm that aims to improve the likelihood of model parameters iteratively, especially in situations where such estimation relies on **unobserved hidden variables** or **missing data** m. This algorithm is called Expectation Maximization (EM).

EM is about trying to fill in the missing variables m using a probabilistic model. It can be applied to any probababilistic model, such as bayesian networks or graphical models, and there is indeed a lot of scientific literature about applying EM algorithms to various models.

The goal of the EM algorithm is to optimize the likelihood of the parameters θ: start from θ_0, improve its likelihood successively until an optimum θ^* is reached:

$$\theta_0 \rightarrow \theta_1 \rightarrow ... \rightarrow \theta_i, \rightarrow \theta_{i+1} \rightarrow ... \rightarrow \theta^* \tag{206}$$

At each step, the change in $\Delta(\theta_{i+1}, \theta_i)$ in log-likelihood is measured:

$$\Delta(\theta_{i+1}, \theta_i) = \ln\ P(D|\theta_{i+1}) - \ln P(D|\theta_i) = \ln\ \frac{P(D|\theta_{i+1})}{P(D|\theta_i)} \tag{207}$$

Optimizing the likelihood then amounts to keeping $\Delta(\theta_{i+1}, \theta_i)$ positive. However the missing data m complicates the computation of the likelihood: instead of having $P(D|\theta)$, usually what we have is $P(D, m|\theta)$. The usual way to deal with m is via marginalization:

- If m is discrete:

$$P(D|\theta) = E_{m|\theta}[P(D, m|\theta)] = \sum_m P(D, m|\theta) = \sum_m P(D|m, \theta)P(m|\theta) \tag{208}$$

- If m is continuous:

$$P(D|\theta) = E_{m|\theta}[P(D, m|\theta)] = \int_m P(D, m|\theta)dm = \int_m P(D|m, \theta)P(m|\theta)dm \tag{209}$$

where $E_{m|\theta}[P(D, m|\theta)]$ is the **expectation** of $P(D, m|\theta)$ over all possible values of m. Note that this is the same as what was done in the forward algorithm where D was the sequence x and m was the path π.

We'll assume that m is discrete from now on for simplification. The measurement of likelihood improvement then becomes:

$$\begin{aligned} \Delta(\theta_{i+1}, \theta_i) &= \ln E[P(D, m|\theta_{i+1})] - \ln P(D|\theta_i) && (210) \\ &= \ln \frac{\sum_m P(D, m|\theta_{i+1})}{P(D|\theta_i)} = \ln \sum_m \frac{P(D, m|\theta_{i+1})}{P(D|\theta_i)} && (211) \end{aligned}$$

Note that only the term for iteration $i+1$ is marginalizing the missing variables m. Moreover since the denominator is not dependent on m, it is ok to push it inside the sum. Anyway the solution of eq. (211) turns out to be difficult mathematically because of the log of a sum. The technique is to use **Jensen's inequality** to get a lower bound δ on Δ. If $\delta > 0$ at each iteration, then we're sure that $\Delta > 0$ too, by only measuring δ.

Jensen's inequality guarantees that the log of a sum is always larger than a weighted sum of logs, for positive weights λ_j that sum up to one:

$$\ln \sum_j \lambda_j y_j \geq \sum \lambda_j \ln y_j \quad \text{for} \quad \lambda_j > 0 \text{ and } \sum_j \lambda_j = 1 \tag{212}$$

Let's multiply eq. (211) by $\lambda_m = P(m|D,\theta_i)$ up and down such that it doesn't change:

$$\begin{aligned}\Delta(\theta_{i+1},\theta_i) &= \ln \sum_m \frac{P(D,m|\theta_{i+1})P(m|D,\theta_i)}{P(D|\theta_i)P(m|D,\theta_i)} &(213)\\ &= \ln \sum_m P(m|D,\theta_i)\frac{P(D,m|\theta_{i+1})}{P(D,m|\theta_i)} &(214)\end{aligned}$$

Now choose $\lambda_m = P(m|D,\theta_i)$ and $y_m = \frac{P(D,m|\theta_{i+1})}{P(D,m|\theta_i)}$ in order to apply Jensen's inequality:

$$\begin{aligned}\Delta(\theta_{i+1},\theta_i) &= \ln \sum_m P(m|D,\theta_i)\frac{P(D,m|\theta_{i+1})}{P(D,m|\theta_i)} &(215)\\ &= \ln \sum_m \lambda_m y_m \geq \sum_m \lambda_m \ln y_m &(216)\\ &= \sum_m P(m|D,\theta_i)\ln\frac{P(D,m|\theta_{i+1})}{P(D,m|\theta_i)} &(217)\\ &= \delta(\theta_{i+1},\theta_i) &(218)\end{aligned}$$

Now instead of trying to maximize Δ directly, we maximize δ, the lower bound on the variation in likelihood:

$$\delta(\theta_{i+1},\theta_i) = \sum_m P(m|D,\theta_i)\ \ln\frac{P(D,m|\theta_{i+1})}{P(D,m|\theta_i)} \tag{219}$$

The goal of the iterative algorithm is to find the parameters θ_{i+1} for the next iteration, given the current θ_i, in such a way that δ is maximized:

$$\begin{aligned}\theta_{i+1}^{EM} &= \underset{\theta}{\text{argmax}}\ \delta(\theta,\theta_i^{EM})\\ &= \underset{\theta}{\text{argmax}}\ \sum_m P(m|D,\theta_i^{EM})\ \ln\frac{P(D,m|\theta)}{P(D,m|\theta_i^{EM})} &(220)\\ &= \underset{\theta}{\text{argmax}}\ \sum_m P(m|D,\theta_i^{EM})\ (\ln P(D,m|\theta) - \ln P(D,m|\theta_i^{EM})) &(221)\\ &= \underset{\theta}{\text{argmax}}\ \sum_m P(m|D,\theta_i^{EM})\ln P(D,m|\theta)\\ &\qquad -\sum_m P(m|D,\theta_i^{EM})\ln P(D,m|\theta_i^{EM}) &(222)\end{aligned}$$

The superscript EM simply means that these are the optimum parameters chosen by the EM algorithm at each iteration.

The last term of eq. (222) is independent of θ so it has no influence on the maximum and can be eliminated:

$$\begin{aligned}\theta_{i+1}^{EM} &= \underset{\theta}{\text{argmax}}\ \sum_m P(m|D,\theta_i^{EM})\ \ln\ P(D,m|\theta)\\ &= \underset{\theta}{\text{argmax}}\ Q_{\theta_i^{EM}}(\theta) &(223)\end{aligned}$$

where

$$Q_{\theta_i^{EM}}(\theta) = E_{m|D,\theta_i}[\ln\ P(D,m|\theta)] = \sum_m P(m|D,\theta_i^{EM})\ \ln\ P(D,m|\theta) \tag{224}$$

is the expectation of $\ln(P(D,m|\theta))$, which is the log-likelihood of the available data D and the missing data m, averaged over all possible values of m, using the current parameter estimation

θ_i^{EM}. Hence maximizing δ amounts to maximizing the expectation function $Q_{\theta_i^{EM}}$. For this reason the algorithm is called Expectation-Maximization.

The algorithm guarantees that the likelihood will increase at each step, because in the worst case we get $\delta = 0$, in which case we simply stick to the θ_i found in the previous iteration: $\theta_{i+1}^{EM} = \theta_i^{EM}$:

$$\delta(\theta_{i+1}^{EM}, \theta_i^{EM}) \geq \delta(\theta_i^{EM}, \theta_i^{EM}) = 0 \quad \Rightarrow \tag{225}$$

$$\ln P(D \mid \theta_{i+1}^{EM}) \geq \ln P(D \mid \theta_i^{EM}) \tag{226}$$

This is our stop critetion, so we stop when δ can no longer increase. So this algorithm is able to find a local maximum, but is not guaranteed to find the global maximum. Figure 11 illustrates this for the oversimplified case where θ has one dimension.

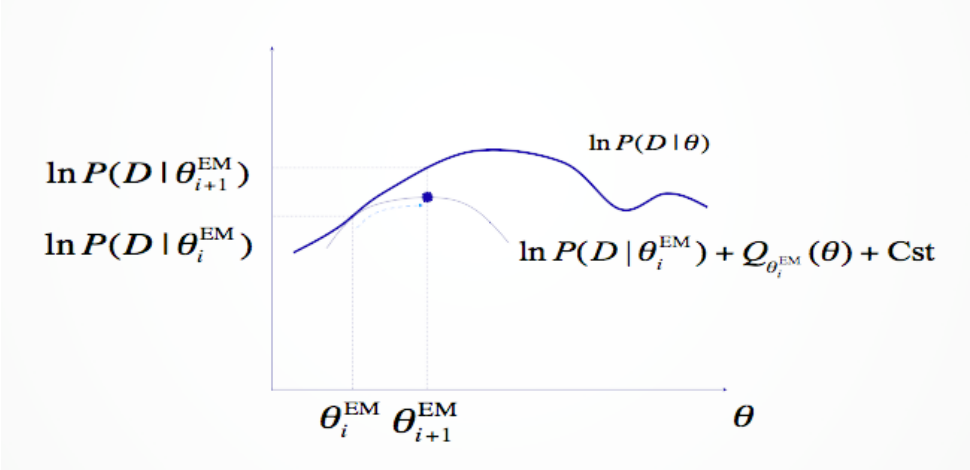

Figure 11: Search for the optimum θ using the Expectation-Maximization Algorithm.

The algorithm works as follows: ok or not ???

- Given: D
- Initialization: set $i = 0$; set θ_0^{EM} to an initial guess
- Repeat until $\delta \approx 0$:

  Expectation step: compute $Q_{\theta_i^{EM}}(\theta)$ using eq. (224) and the parameters θ_i^{EM} from the previous iteration.

  Maximization step: compute θ_{i+1}^{EM} by maximizing $Q_{\theta_i^{EM}}(\theta)$ as shown in eq. (223)

  Update δ using eq. (219)

  $i = i + 1$
- Output θ_i^{EM}

4.1 Baum-Welch algorithm

The Baum-Welch algorithm is a special case of EM where:
$D = x$, $m = \pi$, and $\theta = \{\{a_k l\}, \{e_k(b)\}\}$.

4.2 Motif finding

context: transcription regulation for the expression of genes

enhancer and silencer regions: sites where prots bind, right combination of prots bring RNA pol together or block it to start transcribing the gene

multiple co-regulated genes have a similar TF binding site

simplification: Each DNA sequence contains only one copy of the TF binding site. the motif has fixed length (given).

motif = binding site (not exact match)

motif finding: find the position of the motif in each sequence, without knowing how the motif looks like: we only know that it occurs only once. so the problem is to find the best common motif shared among all sequences (like a local alignment but for multiple sequences)

Prob of seq given the position $Prob(S|a, \Theta_W, \Theta_0)$

$$\ln\ P(s_k|a_k, \theta) = \sum_{i=1}^{a_k-1} \ln\ \theta^0_{b^k_i} + \sum_{i=1}^{w} \ln\ \theta^i_{b^k_{a_k+i-1}} + \sum_{i=a_k+w}^{L_k} \ln\ \theta^0_{b^k_i} \tag{227}$$

for simplification assume that all positions have the same probability of containing the motif:

$$P(a_k|\theta) = \frac{1}{L_k} \tag{228}$$