5

Profile HMMs for sequence families

So far we have concentrated on the intrinsic properties of single sequences, such as CpG islands in DNA, or on pairwise alignment of sequences. However, functional biological sequences typically come in families, and many of the most powerful sequence analysis methods are based on identifying the relationship of an individual sequence to a sequence family. Sequences in a family will have diverged from each other in their primary sequence during evolution, having separated either by a duplication in the genome, or by speciation giving rise to corresponding sequences in related organisms. In either case they normally maintain the same or a related function. Therefore, identifying that a sequence belongs to a family, and aligning it to the other members, often allows inferences about its function.

If you already have a set of sequences belonging to a family, you can perform a database search for more members using pairwise alignment with one of the known family members as the query sequence. To be more thorough, you could even search with all the known members one by one. However, pairwise searching with any one of the members may not find sequences distantly related to the ones you have already. An alternative approach is to use statistical features of the whole set of sequences in the search. Similarly, even when family membership is clear, accurate alignment can be often be improved significantly by concentrating on features that are conserved in the whole family.

How, in brief, do we identify such features? Just as a pairwise alignment captures much of the relationship between two sequences, a multiple alignment can show how the sequences in a family relate to each other. Figure 5.1 shows a multiple alignment of seven sequences from the large globin family (hundreds of globin sequences are available in the protein sequence databases). The three dimensional structure has been obtained for each protein in the alignment shown, and the sequences have been aligned on the basis of aligning the eight alpha helices of the conserved globin fold, and also on the basis of aligning certain key residues in the sequences, such as two conserved histidines (H) which are the residues which interact with an oxygen-binding heme prosthetic group in the globin active site.

It is clear that some positions in the globin alignment are more conserved than others. In general the helices are more conserved than the loop regions between

```
Helix                AAAAAAAAAAAAAAAA   BBBBBBBBBBBBBBBBCCCCCCCCCCC
HBA_HUMAN  ---------VLSPADKTNVKAAWGKVGA--HAGEYGAEALERMFLSFPTTKTYFPHF
HBB_HUMAN  --------VHLTPEEKSAVTALWGKV----NVDEVGGEALGRLLVVYPWTQRFFESF
MYG_PHYCA  ---------VLSEGEWQLVLHVWAKVEA--DVAGHGQDILIRLFKSHPETLEKFDRF
GLB3_CHITP ----------LSADQISTVQASFDKVKG------DPVGILYAVFKADPSIMAKFTQF
GLB5_PETMA PIVDTGSVAPLSAAEKTKIRSAWAPVYS--TYETSGVDILVKFFTSTPAAQEFFPKF
LGB2_LUPLU --------GALTESQAALVKSSWEEFNA--NIPKHTHRFFILVLEIAPAAKDLFS-F
GLB1_GLYDI ---------GLSAAQRQVIAATWKDIAGADNGAGVGKDCLIKFLSAHPQMAAVFG-F
Consensus            Ls....  v a W kv .  .    g . L.. f . P .   F  F

Helix          DDDDDDDEEEEEEEEEEEEEEEEEEE            FFFFFFFFFFFF
HBA_HUMAN  -DLS-----HGSAQVKGHGKKVADALTNAVAHV---D--DMPNALSALSDLHAHKL-
HBB_HUMAN  GDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHL---D--NLKGTFATLSELHCDKL-
MYG_PHYCA  KHLKTEAEMKASEDLKKHGVTVLTALGAILKK----K-GHHEAELKPLAQSHATKH-
GLB3_CHITP AG-KDLESIKGTAPFETHANRIVGFFSKIIGEL--P---NIEADVNTFVASHKPRG-
GLB5_PETMA KGLTTADQLKKSADVRWHAERIINAVNDAVASM--DDTEKMSMKLRDLSGKHAKSF-
LGB2_LUPLU LK-GTSEVPQNNPELQAHAGKVFKLVYEAAIQLQVTGVVVTDATLKNLGSVHVSKG-
GLB1_GLYDI SG----AS---DPGVAALGAKVLAQIGVAVSHL--GDEGKMVAQMKAVGVRHKGYGN
Consensus   .  t    .. . v..Hg kv. a   a...l   d   . a l. l   H  .

Helix       FFGGGGGGGGGGGGGGGGGG     HHHHHHHHHHHHHHHHHHHHHHHHHH
HBA_HUMAN  -RVDPVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR------
HBB_HUMAN  -HVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH------
MYG_PHYCA  -KIPIKYLEFISEAIIHVLHSRHPGDFGADAQGAMNKALELFRKDIAAKYKELGYQG
GLB3_CHITP --VTHDQLNNFRAGFVSYMKAHT--DFA-GAEAAWGATLDTFFGMIFSKM-------
GLB5_PETMA -QVDPQYFKVLAAVIADTVAAG---------DAGFEKLMSMICILLRSAY-------
LGB2_LUPLU --VADAHFPVVKEAILKTIKEVVGAKWSEELNSAWTIAYDELAIVIKKEMNDAA---
GLB1_GLYDI KHIKAQYFEPLGASLLSAMEHRIGGKMNAAAKDAWAAAYADISGALISGLQS-----
Consensus    v.   f  l . ..  ....    f   . aa. k. .     l sky
```

Figure 5.1 *An alignment of seven globins from Bashford, Chothia & Lesk [1987]. To the left is the protein identifier in the* SWISS-PROT *database [Bairoch & Apweiler 1997]. The eight alpha helices are shown as* A–H *above the alignment. A consensus line below the alignment indicates residues that are identical among at least six of the seven sequences in upper case, ones identical in four or five sequences in lower case, and positions where there is a residue identical in three sequences with a dot.*

them, and certain residues are particularly strongly conserved. When identifying a new sequence as a globin, it would be desirable to concentrate on checking that these more conserved features are present. How to obtain and use such information will be the subject of this chapter.

As might be expected, our approach to consensus modelling will be to make a probabilistic model. In particular, we will develop a particular type of hidden Markov model well suited to modelling multiple alignments. We call these *profile HMMs* after standard *profiles*, which are closely related non-probabilistic structures introduced previously for the same purpose by Gribskov, McLachlan & Eisenberg [1987]. Profile HMMs are probably the most popular application of hidden Markov models in molecular biology at the moment [Eddy 1996].

We will assume for the purposes of this chapter that we are given a correct multiple alignment, from which we will build a model that can be used to find and score potential matches to new sequences. The multiple alignment could

be built from structural information, like the globin alignment shown here, or it could come from a sequence-based alignment procedure, such as those discussed in Chapter 6.

Much of this chapter makes use of the theory presented in Chapter 3 for general HMMs. The most important algorithms will be presented again in the specific form relevant to profile HMMs. There is also an extensive discussion of how to estimate optimal probability parameters from multiple sequence alignments.

5.1 Ungapped score matrices

One general feature of protein family multiple alignments, which can be seen in Figure 5.1, is that gaps tend to line up with each other, leaving solid blocks where there are no insertions or deletions in any of the sequences. We will start by considering models for these ungapped regions.

As an example, consider the E helix of Figure 5.1. A natural probabilistic model for such a region would be to specify independent probabilities $e_i(a)$ of observing amino acid a in position i (we use letter e because these will turn out to be the *emission probabilities* of the hidden Markov model when we introduce gaps). The probability of a new sequence x according to this model is then

$$P(x|M) = \prod_{i=1}^{L} e_i(x_i),$$

where L is the length of the block, 21 in this case. As usual, we are in fact more interested in the ratio of this probability to the probability of x under a random model, and so to test for membership in the family we evaluate the log-odds ratio

$$S = \sum_{i=1}^{L} \log \frac{e_i(x_i)}{q_{x_i}}.$$

The values $\log \frac{e_i(a)}{q_a}$ behave like elements in a score matrix $s(a,b)$, where the second index is position i, rather than amino acid b. For this reason, such an approach is known as a *position specific score matrix* (PSSM). A PSSM can be used to search for a match in a longer sequence x of length N by evaluating the score S_j for each starting point j in x from 1 to $N-L+1$, where L is the length of the PSSM.

5.2 Adding insert and delete states to obtain profile HMMs

Although a PSSM captures some conservation information, it is clearly an inadequate representation of all the information in a multiple alignment of a protein

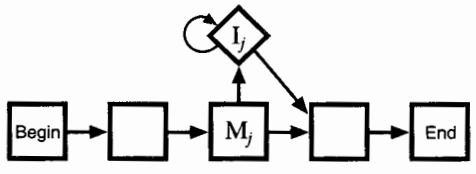
family. We have to find some way to take account of gaps. It is possible to combine the scores of multiple ungapped block models, and this is the approach taken by Henikoff & Henikoff [1991] in the BLOCKS database. However, we will pursue here the aim of developing a single probabilistic model for the whole extent of the alignment.

One approach is to allow gaps at each position in the alignment, using the same gap score $\gamma(g)$ at each position, as in pairwise alignment. However, this is also ignoring information, because the alignment gives us explicit indications of where gaps are more and less likely. We want to capture this information to give us position sensitive gap scores, just as the emission probabilities gave us position sensitive substitution scores.

The approach we take is to build a hidden Markov model (HMM), with a repetitive structure of states, but different probabilities in each position. This will provide a full probabilistic model for sequences in the sequence family. We start off by observing that the PSSM can be viewed as a trivial HMM with a series of identical states that we will call *match* states, separated by transitions of probability 1.

Alignment is trivial because there is no choice of transitions. We rename the emission probabilities for the match states to $e_{M_i}(a)$.

The next step is to deal with gaps. We must treat insertions and deletions separately. To handle insertions, i.e. portions of x that do not match anything in the model, we introduce a set of new states I_i, where I_i will be used to match insertions after the residue matching the ith column of the multiple alignment. The I_i have emission distribution $e_{I_i}(a)$, but these are normally set to the background distribution q_a, just as for seeing an unaligned inserted residue in a pairwise alignment. We need transitions from M_i to I_i, a loop transition from I_i to itself, to accommodate multi-residue insertions, and a transition back from I_i to M_{i+1}. Here is a single insert state of this kind:

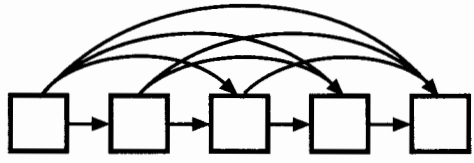


We denote insert states in our diagrams by diamonds. The log-odds cost of an insert is the sum of the costs of the relevant transitions and emissions. Assuming that $e_{I_i}(a) = q_a$ as described above, there is no log-odds contribution from the emission, and the score of a gap of length k is

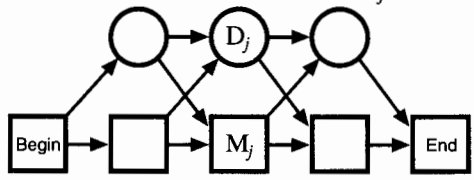
$$\log a_{M_j I_j} + \log a_{I_j M_{j+1}} + (k-1)\log a_{I_j I_j}.$$

From this you can see that the type of insert state shown corresponds to an affine gap scoring model.

Deletions, i.e. segments of the multiple alignment that are not matched by any residue in x, could be handled by forward 'jump' transitions between non-neighbouring match states:



However, to allow arbitrarily long gaps in a long model this way would require a lot of transitions. Instead we introduce silent states D_j as described in Section 3.4:



Because the silent states do not emit any residues, it is possible to use a sequence of them to get from any match state to any later one, between two residues in the sequence. The cost of a deletion will then be the sum of the costs of an M $\rightarrow$ D transition followed by a number of D $\rightarrow$ D transitions, then a D $\rightarrow$ M transition. This is at first sight exactly analogous to the cost of an insert, although the path through the model looks different. In detail, it is possible that the D $\rightarrow$ D transitions will have different probabilities, and hence contribute differently to the score, whereas all the I $\rightarrow$ I transitions for one insert involve the same state, and so are guaranteed to have the same cost.

The full resulting HMM has the structure shown in Figure 5.2. This form of model, which we call a profile HMM, was first introduced in Haussler *et al.* [1993] and Krogh *et al.* [1994]. We have added transitions between insert and delete states, as they did, although these are usually very improbable. Leaving them out has negligible effect on scoring a match, but can create problems when building the model.

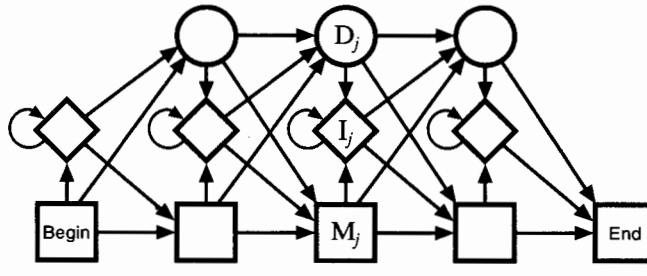


Figure 5.2 *The transition structure of a profile HMM. We use diamonds to indicate the insert states and and circles for the delete states.*

Profile HMMs generalise pairwise alignment

We have seen how the costs of using gap states in a profile HMM mirror those used in pairwise alignment with affine gaps. To help make clear the relationship, it is useful to consider the degenerate case where the multiple alignment from which we build the HMM contains just one sequence.

Let us compare Figure 5.2 with Figure 4.2. If we call the example sequence y, then Figure 5.2 is an unrolled version of Figure 4.2, with the y_j emissions each coming from a separate copy of the pair HMM. The states M_j correspond to a sequence of match states M, the I_j to corresponding incarnations of X, and the D_j to incarnations of Y. To achieve as close a correspondence as possible, the natural values for the match emission probabilities $e_{\mathrm{M}_i}(a)$ are $p_{y_i a}/q_{y_i}$, the conditional probabilities of seeing a given y_i in a pairwise alignment, and for the transition probabilities $a_{\mathrm{M}_i\mathrm{I}_i} = a_{\mathrm{M}_i\mathrm{D}_{i+1}} = \delta$ and $a_{\mathrm{I}_i\mathrm{I}_i} = a_{\mathrm{D}_i\mathrm{D}_{i+1}} = \varepsilon$ for all i.

In formal terms our profile HMM is effectively the hidden Markov model obtained by conditioning the pair HMM of Figure 4.2 on emitting sequence y as one of the sequences in its alignment. Because of this, the Viterbi equations for finding the most probable alignment of x to our profile HMM are essentially the same as those for the most probable alignment of x and y to the pair HMM described in Chapter 4. If we convert them into log-odds ratio form we recover our standard affine gap cost pairwise alignment equations of (2.16), as we will see below. Any differences are due to slightly different Begin and End arrangements.

5.3 Deriving profile HMMs from multiple alignments

Although it is nice to see that the profile HMM is doing the same sort of dynamic programming as we have used before for pairwise alignment, this is not why we introduced them. The key idea behind profile HMMs is that we can use the same structure as shown in Figure 5.2, but set the transition and emission probabilities to capture specific information about each position in the multiple alignment of the whole family. Essentially, we want to build a model representing the consensus sequence for the family, not the sequence of any particular member.

There are a number of different ways to derive the parameter values from a multiple alignment of the sequences in the family. To provide an example for illustrating these methods, Figure 5.3 shows a short section of the globin alignment shown in Figure 5.1.

Non-probabilistic profiles

A model similar to the profile HMM was first introduced by Gribskov, McLachlan & Eisenberg [1987] who coined the name 'profile' (see also Gribskov, Lüthy & Eisenberg [1990]). However, they did not have an underlying probabilistic model,

```
HBA_HUMAN    ...VGA--HAGEY...
HBB_HUMAN    ...V----NVDEV...
MYG_PHYCA    ...VEA--DVAGH...
GLB3_CHITP   ...VKG------D...
GLB5_PETMA   ...VYS--TYETS...
LGB2_LUPLU   ...FNA--NIPKH...
GLB1_GLYDI   ...IAGADNGAGV...
                  ***  *****
```

Figure 5.3 *Ten columns from the multiple alignment of seven globin protein sequences shown in Figure 5.1. The starred columns are ones that will be treated as 'matches' in the profile HMM.*

but rather directly assigned position specific scores for each match state and gap penalty, for use in standard 'best match' dynamic programming. They set the scores for each consensus position to the averages of the standard substitution scores from all the residues seen in the corresponding multiple alignment column. For example, they would set the score for residue a in column 1 of our example to be

$$\tfrac{5}{7}s(\mathrm{V},a)+\tfrac{1}{7}s(\mathrm{F},a)+\tfrac{1}{7}s(\mathrm{I},a)$$

where $s(a,b)$ is the standard substitution matrix. They also set gap penalties for each column using a heuristic equation that decreased the cost of a gap (either insertion or deletion) according to the length of the longest gap observed in the multiple alignment spanning the column.

Although this seems an intuitively obvious way to combine information, and it has been used effectively by many people for finding new members of families, it does produce anomalies. For example, column 1 is much more strongly conserved than column 2 in the example shown in Figure 5.3, but the information in column 1 will be smeared out just as much by the substitution matrix as that in column 2. If we had an alignment with 100 sequences, all with a cysteine (C) at some position, then the implicit probability distribution for that column for an 'average' profile would be exactly the same as would be derived from a single sequence. This does not correspond to our expectation that the likelihood of a cysteine should go up as we see more confirming examples.

In addition to these observations about substitution scores, the scores for gaps do not behave as expected. For example, from the alignment in Figure 5.3 the score for a deletion would be set to be the same in column 2, where there is a deletion in one sequence, HBB_HUMAN, as in column 4, where there is a deletion opening in five of the seven sequences. It would be more reasonable to set the probability of a new gap opening to be higher in column 4.

Changes have been made to non-probabilistic profiles to address these and

other problems [Thompson, Higgins & Gibson 1994b; Gribskov & Veretnik 1996], and we shall return to some of these later.

Basic profile HMM parameterisation

Let us turn back to hidden Markov model profiles. Like all HMMs, these have emission and transition probabilities. Assuming that these probabilities are non-zero, a profile HMM can model any possible sequence of residues from the given alphabet. It therefore defines a probability distribution over the whole space of sequences. The aim of the parameterisation process it to make this distribution peak around members of the family.

The parameters we have available to control the shape of the distribution are the values of the probabilities, and also the length of the model. There is a lot to say about setting these optimally. We give here the basic methods from Krogh *et al.* [1994]. After sections on database searching and variants for local alignment, we will return to an extended discussion of alternative parameter estimation techniques.

The choice of length of the model corresponds more precisely to a decision on which multiple alignment columns to assign to match states, and which to assign to insert states. The profile HMM we derived above from the single sequence y had a match state for each residue y_i. However, looking at Figure 5.3 it seems clear that the consensus sequence for this region should only have eight residues. and that the two non-starred residues in `GLB1_GLYDI` should be treated as an insertion with respect to the consensus. For the time being we will use a heuristic rule to decide which columns should correspond to match states, and which to inserts. A simple rule that works well is that columns that are more than half gap characters should be modelled by inserts.

The second problem is how to assign the probability parameters. We regard the alignment as providing a set of independent samples of alignments of sequences x to our HMM. Since the alignments are given, we can estimate the parameters directly using equations (3.18) from Section 3.3. We just count up the number of times each transition or emission is used, and assign probabilities according to

$$a_{kl} = \frac{A_{kl}}{\sum_{l'} A_{kl'}} \quad \text{and} \quad e_k(a) = \frac{E_k(a)}{\sum_{a'} E_k(a')}$$

where k and l are indices over states, and a_{kl} and e_k are the transition and emission probabilities, and A_{kl} and E_k are the corresponding frequencies.

In the limit of having a very large number of sequences in our training alignment, this will give an accurate and consistent estimate of the probabilities. However, it has problems when there are only a few sequences. A major difficulty is that some transitions or emissions may not be seen in the training alignment, and so would acquire zero probability, which would mean they would never be

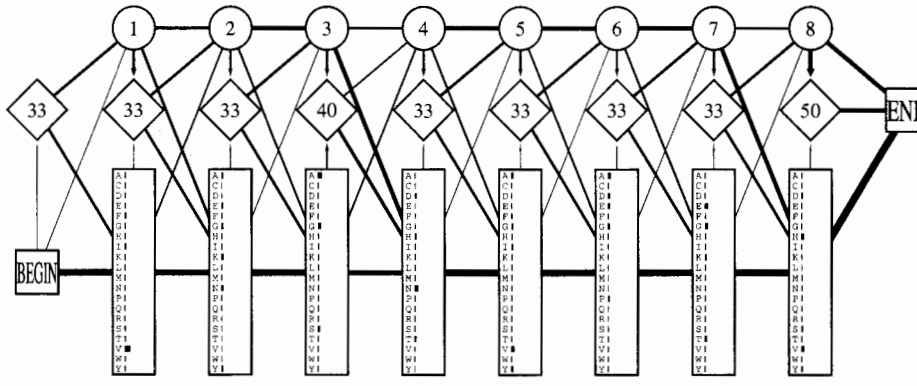


Figure 5.4 *A hidden Markov model derived from the small alignment shown in Figure 5.3 using Laplace's rule. Emission probabilities are shown as bars opposite the different amino acids for each match state, and transition probabilities are indicated by the thickness of the lines. The* $\mathrm{I} \rightarrow \mathrm{I}$ *transition probabilities times 100 are shown in the insert states. (Figure generated automatically using the* SAM *package.)*

allowed in the future. More broadly, we are not using any previous knowledge about protein alignments, as the earlier non-probabilistic methods did implicitly, by using an independently derived substitution matrix. As a minimal approach to avoid zero probabilities, we can add pseudocounts to the observed frequencies (as in Chapters 1 and 3). The simplest pseudocount method is Laplace's rule: to add one to each frequency. We discuss better ways to choose the pseudocount values, and other approaches to estimating the parameters, at greater length below in Section 5.6.

Example: Parameters for an HMM based on Figure 5.3

Let us assume that we use Laplace's rule to obtain parameters for an HMM corresponding to the alignment in Figure 5.3. Then $e_{\mathrm{M}_1}(\mathtt{V}) = 6/27, e_{\mathrm{M}_1}(\mathtt{I}) = e_{\mathrm{M}_1}(\mathtt{F}) = 2/27$, and $e_{\mathrm{M}_1}(a) = 1/27$ for all residue types a other than V, I, F. Similarly, $a_{\mathrm{M}_1\mathrm{M}_2} = 7/10, a_{\mathrm{M}_1\mathrm{D}_2} = 2/10$ and $a_{\mathrm{M}_1\mathrm{I}_1} = 1/10$ (following column 1 there are six transitions from match to match, one transition to a delete state, in HBB_HUMAN, and no insertions). Figure 5.4 shows the complete set of parameters for the HMM in diagrammatic form. □

5.4 Searching with profile HMMs

One of the main purposes of developing profile HMMs is to use them to detect potential membership in a family by obtaining significant matches of a sequence to the profile HMM. We will assume for now that we are looking for global matches.

In practice, as for pairwise alignment, one of the local alignment methods may be more sensitive for finding distant matches. We discuss these in the next section.

We have a choice of ways to score a match to a hidden Markov model. We can either use the Viterbi equations to give the most probable alignment π^* of a sequence x together with its probability $P(x,\pi^*|M)$, or the forward equations to calculate the full probability of x summed over all possible paths $P(x|M)$.

In either case, for practical purposes the result we want to consider when evaluating potential matches is the log-odds ratio of the resulting probability to the probability of x given our standard random model

$$P(x|R) = \prod_i q_{x_i}.$$

We therefore show here versions of the Viterbi and forward algorithms that are designed specifically for profile HMMs, and which result directly in the desired log-odds values. Note that changing to log-odds does not change the result; we could have subtracted the random model log score afterwards. However, it is cleaner and more efficient. Another practical reason for working in log-odds units is to avoid problems of underflow when working with raw probabilities, as we discussed in Section 3.6.

Viterbi equations

Let $V_j^{\mathrm{M}}(i)$ be the log-odds score of the best path matching subsequence $x_{1\ldots i}$ to the submodel up to state j, ending with x_i being emitted by state M_j. Similarly $V_j^{\mathrm{I}}(i)$ is the score of the best path ending in x_i being emitted by I_j, and $V_j^{\mathrm{D}}(i)$ for the best path ending in state D_j. Then we can write

$$\begin{aligned}
V_j^{\mathrm{M}}(i) &= \log \frac{e_{\mathrm{M}_j}(x_i)}{q_{x_i}} + \max \begin{cases} V_{j-1}^{\mathrm{M}}(i-1) + \log a_{\mathrm{M}_{j-1}\mathrm{M}_j}, \\ V_{j-1}^{\mathrm{I}}(i-1) + \log a_{\mathrm{I}_{j-1}\mathrm{M}_j}, \\ V_{j-1}^{\mathrm{D}}(i-1) + \log a_{\mathrm{D}_{j-1}\mathrm{M}_j}; \end{cases} \\
V_j^{\mathrm{I}}(i) &= \log \frac{e_{\mathrm{I}_j}(x_i)}{q_{x_i}} + \max \begin{cases} V_j^{\mathrm{M}}(i-1) + \log a_{\mathrm{M}_j\mathrm{I}_j}, \\ V_j^{\mathrm{I}}(i-1) + \log a_{\mathrm{I}_j\mathrm{I}_j}, \\ V_j^{\mathrm{D}}(i-1) + \log a_{\mathrm{D}_j\mathrm{I}_j}; \end{cases} \qquad (5.1) \\
V_j^{\mathrm{D}}(i) &= \max \begin{cases} V_{j-1}^{\mathrm{M}}(i) + \log a_{\mathrm{M}_{j-1}\mathrm{D}_j}, \\ V_{j-1}^{\mathrm{I}}(i) + \log a_{\mathrm{I}_{j-1}\mathrm{D}_j}, \\ V_{j-1}^{\mathrm{D}}(i) + \log a_{\mathrm{D}_{j-1}\mathrm{D}_j}. \end{cases}
\end{aligned}$$

These are the general equations. In a typical case, there is no emission score $e_{\mathrm{I}_j}(x_i)$ in the equation for $V_j^{\mathrm{I}}(i)$ because we assume that the emission distribution from the insert states I_j is the same as the background distribution, so the probabilities cancel in the log-odds form. Also, the $\mathrm{D} \rightarrow \mathrm{I}$ and $\mathrm{I} \rightarrow \mathrm{D}$ transition terms may not be present, as discussed above.

We need to take a little care over initialisation and termination of the dynamic programming. We want to allow the alignment to start and end in a delete or insert state, in case the beginning or end of the sequence does not match the first or the last match state of the model. The simplest way to ensure this mechanistically is to rename the Begin state as M_0 and set $V_0^M(0) = 0$ (as we did in Chapter 3). We then allow transitions to I_0 and D_1. Similarly, at the end we can collect together possible paths ending in insert and delete states by renaming the End state to M_{L+1} and using the top relation without the emission term to calculate $V_{L+1}^M(n)$ as the final score.

If these recurrence equations are compared with those for standard gapped dynamic programming in (2.16), it can be seen that apart from renaming of variables this is the same algorithm, but with the substitution, gap-open and gap-extend scores all depending on position in the model, j.

Forward algorithm

The recurrence equations for the forward algorithm are similar to the Viterbi equations, but with the max() operation replaced by addition. We define variables $F_j^M(i), F_j^I(i)$ and $F_j^D(i)$ for the partial full log-odds ratios, corresponding to $V_j^M(i), V_j^I(i)$ and $V_j^D(i)$. The recurrence equations are then:

$$\begin{aligned}
F_j^M(i) &= \log \frac{e_{M_j}(x_i)}{q_{x_i}} + \log\big[a_{M_{j-1}M_j} \exp\big(F_{j-1}^M(i-1)\big) \\
&\quad + a_{I_{j-1}M_j} \exp\big(F_{j-1}^I(i-1)\big) + a_{D_{j-1}M_j} \exp\big(F_{j-1}^D(i-1)\big)\big]; \\
F_j^I(i) &= \log \frac{e_{I_j}(x_i)}{q_{x_i}} + \log\big[a_{M_jI_j} \exp\big(F_j^M(i-1)\big) \\
&\quad + a_{I_jI_j} \exp\big(F_j^I(i-1)\big) + a_{D_jI_j} \exp\big(F_j^D(i-1)\big)\big]; \\
F_j^D(i) &= \log\big[a_{M_{j-1}D_j} \exp\big(F_{j-1}^M(i)\big) + a_{I_{j-1}D_j} \exp\big(F_{j-1}^I(i)\big) \\
&\quad + a_{D_{j-1}D_j} \exp\big(F_{j-1}^D(i)\big)\big].
\end{aligned}$$

Initialisation and termination conditions are handled as for the Viterbi case, with $F_0^M(0)$ being initialised to 0.

Although these appear a little complicated, in a practical implementation the operation $\log(e^x + e^y)$ can be performed efficiently to adequate accuracy by function lookup and interpolation; see Section 3.6.

Alternatives to log-odds scoring

In some of the earlier papers on HMMs, rather than calculating the log-odds score relative to a random model, the logarithm of the probability of the sequence given the model was used directly. This was called the LL score for 'log likelihood': $\mathrm{LL}(x) = \log P(x|M)$. The LL score is strongly length dependent, so for searching

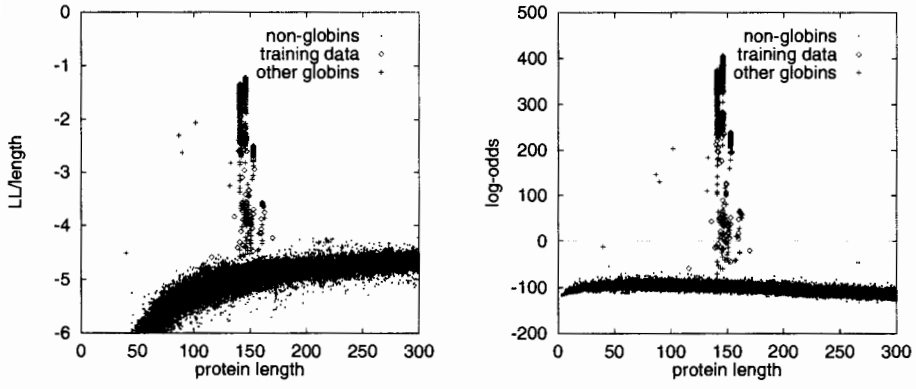


Figure 5.5 *To the left the length-normalized LL score is shown as a function of sequence length. The right plot shows the same for the log-odds score.*

it is not good enough to use a simple threshold. It is better to use LL divided by the sequence length, but even that is not always perfect, because the dependence between LL and sequence length is not linear (see example below).

A way to get around this is to estimate an average score and a standard deviation as a function of length and then use the number of standard deviations each sequence is away from the average. This is called the Z-score, and is also illustrated in the example below.

Example: Modelling and searching for globins

From 300 randomly picked globin sequences a profile HMM was estimated from scratch, i.e. starting from unaligned sequences using procedures we will explain in Chapter 6. A simple pseudocount regulariser was used. The estimation was done several times and the model with the highest overall LL score was picked. (We used the default settings of the SAM package, version 1.2; Hughey & Krogh [1996]).

With this model a database of about 60 000 proteins (SWISS-PROT release 34; Bairoch & Apweiler [1997]) was searched using the forward algorithm. The LL and log-odds scores were found for each sequence. For the null model we used the amino acid frequencies of the 300 sequences in the training set. In Figure 5.5 the length-normalised scores are shown for all the globins in the training set, all the other globins in the database and all the rest of the proteins with lengths up to 300 amino acids.[1] The globin sequences are clearly separated from the non-globins apart from a few in the 'twilight zone.'

The main difference between the two is in the variance of the score for non-globins, which is lower for the log-odds score, and therefore the separation is clearer. However, just choosing a cut-off of zero for the log-odds would miss a

[1] A few dubious globins and other strange sequences were removed from these data.

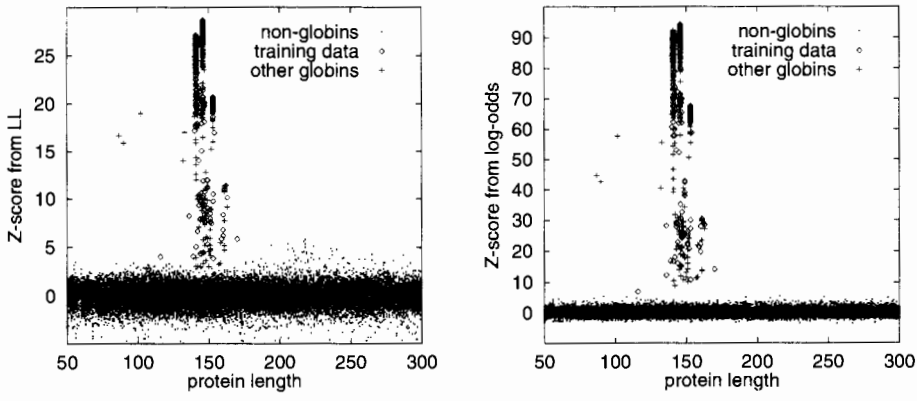


Figure 5.6 *The Z-score calculated from the LL scores (left) and the log-odds (right).*

lot of real globins in the search. This is because the profile HMM is not broad enough: it is too concentrated on a subset of the globins. Although there are ways to address this problem directly that we will return to later in the chapter, it is also possible to take a pragmatic approach to the separation of signal from noise given the results of the search, and calculate Z-scores for each hit.

To calculate Z-scores, a smooth curve is fitted to the LL or log-odds score of the non-globin sequences (a method is outlined in Krogh *et al.* [1994]). A standard deviation is then estimated for each length (or rather a little interval around it), and for each score the distance from the smooth curve is calculated in units of the standard deviation. This is the Z-score. The result (still as a function of sequence length) is shown in Figure 5.6.[2]

It is evident that it is now possible to find a threshold which will separate most globins from all other sequences. It is also clear that the score based on log-odds is much better for discrimination, with approximately three times the signal to noise ratio of the LL score. The reason for this is that dividing by the probability of the random model adjusts for the residue composition of the sequence. Without doing that, sequences with similar residue compositions as globins will tend to score more highly than sequences containing different residues, increasing the variance of the noise. □

Alignment

Aside from finding matches, the other principal use of profile HMMs is to give an alignment of a sequence to the family, or more precisely to add it into the multiple alignment of the family. This is primarily the subject of the next chapter,

[2] There is no analytical result about the shape of these score distributions. The global alignment distribution is probably not exactly a Gaussian [Waterman 1995], but it appears to be a good approximation. For local alignments the extreme value distribution may be more reasonable, as discussed in Chapter 2.

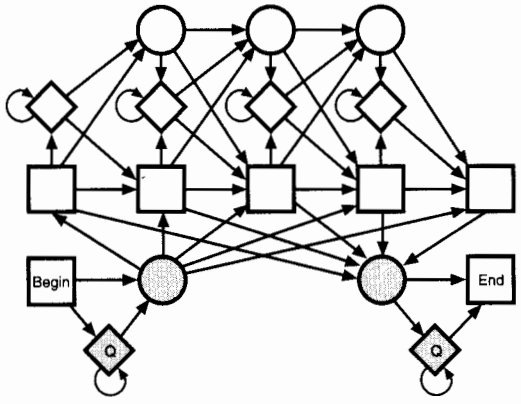
on multiple alignment methods, which covers alignment with profile HMMs at length. For now, we will just point out that the natural solution is to take the highest scoring, or Viterbi, alignment. This is obtained by tracing back on the Viterbi variables $V_j^{\bullet}(i)$, exactly as with pairwise alignment. Beyond this, all the methods of Chapter 4 can be applied, to explore variants, and to assess the reliability of the alignment.

5.5 Profile HMM variants for non-global alignments

We have seen that there is a very close relationship between the Viterbi alignment of a sequence to a profile HMM and the global dynamic programming comparison between two sequences using affine gap penalties, which we described in Chapter 2. It is therefore possible to generalise all the variations of dynamic programming, such as those that find local, repeat and overlap matches, to use profile HMMs.

However, we have developed probabilistic models much more fully since Chapter 2, and this time we want to take more care to ensure that the result of converting to a local algorithm remains a proper probabilistic model, i.e. that we assign each sequence a true probability so that the sum over all sequences $\sum_x P(x|M) = 1$. Our approach to doing this is to specify a new model for the complete sequence x, which incorporates the original profile HMM together with one or more copies of a simple self-looping model that is used to account for the regions of unaligned sequence. These behave very like the insert states that we added to the profile itself. We call them *flanking model* states, because they are used to model the flanking sequences to the actual profile match itself.

The model for local (Smith–Waterman style) alignment is shown here:

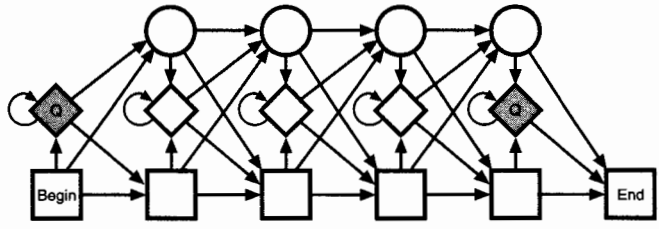


The flanking model states are shown as shaded diamonds. Notice that as well as specifying the emission probabilities of the new states, which will normally of course be q_a, we must specify a number of new transition probabilities. The looping probability on the flanking states should be close to 1, since they must

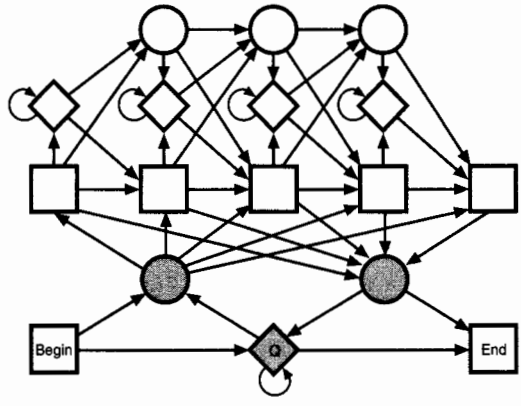
account for long stretches of sequence. Let us set these to $(1 - \eta)$. Note also that we have made use of silent states, shown as shaded circles, as 'switching points' to reduce the total number of transitions.

The next issue is how to set all the transition probabilities from the left flanking state to different start points in the model. One option is to give them equal probabilities, η/L. Another is to assign more probability to starting at the beginning of the model. The default option in the HMMER package for profile HMMs [Eddy 1996] assigns probability $\eta/2$ to the start of the profile, and $\eta/(2(L-1))$ to the other positions, favouring matches that start at the beginning of the model.

If all the probability is assigned to the first model state, then it forces this model to match only complete copies of the profile in the searched sequence, ensuring a type of 'overlap' match constraint. This can be appropriate when, for example, the HMM represents a protein domain that you expect to find either present as a whole or absent. However, to allow for rare cases where the first residue might be missing, it may be wise in such cases to allow a direct transition from the flanking state into a delete state, as shown here:



It is clear that by tinkering with the transition connections and probabilities a wide variety of different models can be produced, each potentially useful in different circumstances. A final example similar to the first model for local matches is



which allows repeat matches to subsections of the profile model, like the repeat algorithm variant in Chapter 2.

Note that all these variants of transition connectivity and probability assignment affect not only the types of match that are allowed, but also the score. More

restrictive transition distributions will give higher match scores if a good match is found, so are preferable if they can be designed to represent the types of correct matches that are expected.

Exercises

5.1 Show that if the random model is the same as that described in Chapter 4 (a succession of two states looping on themselves with probability $(1 - \eta)$), with η the same as in the flanking models, the local alignment model gives update equations like those of equation (2.9).

5.2 Explain the reasons for any differences.

5.6 More on estimation of probabilities

As promised above, we now return to the subject of parameter estimation at greater length. Although our discussion for most of this section will be focused on the emission probabilities, analogous methods can be used for the transition probabilities. The aim here is to introduce methods that can be used. A more detailed mathematical discussion about the estimation of probabilites from sample counts is given in Chapter 11 (p. 311).

The most straightforward approach to parameter estimation would be to give the maximum likelihood estimates for the parameters. We will change notation slightly from that used before. Given observed frequencies c_{ja} of residue a in position j of the alignment, maximum likelihood estimates for $e_{\mathrm{M}_j}(a)$, the corresponding model parameters, are

$$e_{\mathrm{M}_j}(a) = \frac{c_{ja}}{\sum_{a'} c_{ja'}}. \tag{5.2}$$

As we described above, a clear problem with this is that if there are no observed examples of a particular outcome then its probability is estimated as zero. This will frequently occur. For example, in the alignment of Figure 5.3 only `V`, `I` and `F` are present in the first column. However, it is quite likely that other amino acids will occur in that position amongst all the other globin sequences in biology. The easiest way to deal with this problem is to add pseudocounts to the observed counts c_{ja}. Below, we first discuss the pseudocount approach at greater length, then give some more complex alternatives.

Simple pseudocounts

A very simple and much-used pseudocount method is to add a constant to all the counts, which prevents the problem with zero probabilities. When the constant is one, as we used above in our example, this is called 'Laplace's rule'. A slightly

more sophisticated method is to add a quantity proportional to the background distribution, giving

$$e_{M_j}(a) = \frac{c_{ja} + Aq_a}{\sum_{a'} c_{ja'} + A}, \tag{5.3}$$

where c_{ja} are the real counts, and A is the weight put on the pseudocounts as compared to the real counts. Values of A of around twenty seem to work well for protein alignments.

This form of regularisation has the appealing feature that $e_{M_j}(a)$ is approximately equal to q_a if very little data is available, i.e. all the real counts are very small compared to A. At the other extreme, where a large amount of data is available, the effect of the regulariser becomes insignificant and $e_{M_j}(a)$ is essentially equal to the maximum likelihood solution. So, at this intuitive level, pseudocounts make a lot of sense.

Adding pseudocounts amounts to adding some fake imagined data into the alignment, based on our general knowledge of proteins, to represent all the other things that might happen. They thus correspond to prior information about protein families, before having seen the specific data for the family in the form of the alignment. This statement can be formalised in a Bayesian framework. Bayes' equation tells us how to combine data, D, with a prior probability distribution over the parameters $P(\theta)$ to give a posterior distribution over θ, from which we can take either the maximum or the mean as our best estimate,

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}.$$

In our case the parameters θ are our model probabilities. The pseudocount method given above corresponds in this Bayesian framework to assuming a Dirichlet prior distribution with parameters $\alpha_a = Aq_a$ over the probabilities; see Chapter 11 for mathematical details.

Dirichlet mixtures

The problem with the simple pseudocounts, as compared to the substitution matrix based methods, is that only the most rudimentary prior knowledge can be contained in a single pseudocount vector. For this reason we need a lot of example data in the alignment to get good estimates of the parameters. Experience suggests that to achieve good discrimination typically fifty or more examples are desirable when modelling proteins.

In order to include better prior information, it was therefore suggested by Brown *et al.* [1993] that one should use a *mixture* of Dirichlet distributions as the prior. The idea is that there might be several different sets of pseudocount priors $\alpha_\bullet^1, \ldots, \alpha_\bullet^K$ corresponding to different types of alignment environments, where

α_a^k corresponds to Aq_a in the example above. One set might be relevant for exposed loop environments, one for buried small residue environments, etc. Given our counts c_{ja} we first estimate how likely each prior distribution k is (based on how well it fits the observed data), then combine their effects according to these posterior probabilities:

$$e_{M_j}(a) = \sum_k P(k|\boldsymbol{c}_j) \frac{c_{ja} + \alpha_a^k}{\sum_{a'}(c_{ja'} + \alpha_{a'}^k)},$$

where the $P(k|\boldsymbol{c}_i)$ are the *posterior mixture coefficients*. We calculate these by Bayes' rule,

$$P(k|\boldsymbol{c}_j) = \frac{p_k P(\boldsymbol{c}_j|k)}{\sum_{k'} p_k' P(\boldsymbol{c}_j|k')}$$

where the p_k are the prior probabilities of each mixture component, and $P(\boldsymbol{c}_j|k)$ is the probability of the data according to Dirichlet mixture k. The equation for $P(\boldsymbol{c}_j|k)$ has a frightening looking form, which is in fact fairly simple to calculate:

$$P(\boldsymbol{c}_j|k) = \frac{\left(\sum_a c_{ja}\right)!}{\prod_a c_{ja}!} \frac{\Gamma\left(\sum_a c_{ja} + \alpha_a^k\right)}{\prod_a \Gamma(c_{ja} + \alpha_a^k)} \frac{\Gamma\left(\sum_a \alpha_a^k\right)}{\prod_a \Gamma(\alpha_a^k)},$$

where $\Gamma(x)$ is the gamma function, a standard function over the reals related to the factorial function on the integers. For further details and an explanation of this equation, see Chapter 11, where we also describe how the mixture component distributions $\alpha_\bullet^k$ are obtained.

Using this type of approach, it seems that good profile HMMs can be fit to alignments with as few as ten or twenty examples [Sjölander *et al.* 1996].

Substitution matrix mixtures

An alternative approach to using a mixture of Dirichlets is to adjust the pseudocounts in a single Dirichlet formulation, using information from the observed counts and a substitution matrix. This is not a theoretically well-founded approach, but it makes intuitive sense as a heuristic, combining features of the non-probabilistic profile methods and the Dirichlet pseudocount methods.

The first step is to convert the matrix entries $s(a,b)$ into conditional probabilities $P(b|a)$. If we assume that the substitution matrix entries are derived as log-odds ratios, as in Chapter 2, then $s(a,b) = \log(P(a,b)/q_a q_b)$, which is the same as $\log(P(b|a)/P(b))$, so $P(b|a) = q_b e^{s(a,b)}$. We can in fact derive $P(b|a)$ values from an arbitrary score matrix $s(a,b)$ given background probabilities q_a; see below.

Given conditional probabilities $P(b|a)$ we can generate pseudocounts as follows. Let f_{ja} be the maximum likelihood probabilities derived from the counts,

so $f_{ja} = c_{ja} / \sum_{a'} c_{ja'}$. Using these we set pseudocount values with

$$\alpha_{ja} = A \sum_b f_{jb} P(a|b),$$

where A is a positive constant comparable to the one we used with simple pseudocounts [Tatusov, Altschul & Koonin 1994; Claverie 1994; Henikoff & Henikoff 1996]. We then use essentially the same equation as (5.3) to obtain the model parameters:

$$e_{M_j}(a) = \frac{c_{ja} + \alpha_{ja}}{\sum_{a'} c_{ja'} + \alpha_{ja'}}.$$

There is no obvious statistical interpretation for this type of pseudocount, but the idea is quite natural: amino acid i contributes to pseudocount j in proportion to its abundance in the column and the probability of its changing to amino acid j. The formula interpolates between the treatment of pairwise alignments and the maximum likelihood solution. The substitution matrix term dominates if there are small numbers of sequences (especially if $A \gg 1$), and values close to the maximum likelihood estimate are obtained when the number of counts is large (more precisely when the total number of counts $C_j \gg A$).

There are various choices for the scaling constant A of the pseudocounts. For instance $A = 1$ was used in Lawrence *et al.* [1993], but this appears to be too weak in practice. Claverie [1994] suggests $A = \min(20, N)$, and Henikoff & Henikoff [1996] suggest $A = 5R$, where R is the number of different residue types observed in the column (i.e. the number of a for which $c_{ja} > 0$).

Deriving $P(b|a)$ from an arbitrary matrix

Even if a score matrix $s(a,b)$ was not derived as a log-odds matrix, as long as certain conditions are fulfilled it is possible to find a scale factor λ such that $\lambda s(a,b)$ will behave correctly when interpreted as a log-odds matrix [Altschul 1991]. The conditions are that the matrix is negatively biased, i.e. $\sum_{ab} q_a q_b s(a,b) < 0$, and that it contains at least one positive entry.

What we want is a set of values r_{ij} for which

$$s(a,b) = \frac{1}{\lambda} \log \frac{r_{ab}}{q_a q_b},$$

where r_{ab} can be interpreted as the probability for the pair a,b. This equation is easily inverted, so we get the pair probabilities expressed in terms of the substitution matrix $r_{ab} = q_a q_b \exp(\lambda s(a,b))$. To be legitimate probabilities the r_{ab} have to sum to one. We therefore need to find a λ such that

$$f(\lambda) = \sum_{a,b} q_a q_b e^{\lambda s(a,b)} - 1 = 0. \tag{5.4}$$

One such value is $\lambda = 0$, but clearly this is not what we want. The two conditions

we gave above turn out to be sufficient to ensure there is another, positive solution to this equation; see the exercises below.

The resulting value of λ is called the natural scaling factor of the substitution matrix. This probabilistic interpretation of the substitution matrix leads to an entropy measure for the matrix of $\sum_{ab} r_{ab} \log(r_{ab}/q_a q_b)$, which is a useful quantity for characterising and comparing substitution matrices [Altschul 1991].

Exercises

5.3 Use the negative bias condition to show that $f(\lambda)$ is negative for small enough λ. Hint: calculate $f'(0)$, the derivative of $f(\lambda)$ at $\lambda = 0$.

5.4 Use the second condition, that there is at least one positive $s(a,b)$, to show that $f(\lambda)$ becomes positive for large enough λ.

5.5 Finally, show that the second derivative of $f(\lambda)$ is positive, and from this and the results of the previous two exercises that there is one and only one positive value of λ satisfying (5.4).

Estimation based on an ancestor

There is a more principled and direct way to use the information in substitution matrices for estimating the HMM probabilities than that described above. This approach does not use pseudocounts. Instead, it assumes that all the observed sequences have been derived independently from a common ancestor, and generates an estimate of the residue present in a given position in that common ancestor (or rather a posterior probability distribution for what that residue was). From this we can estimate the probability of seeing each residue in a new descendant of the ancestor, different from those in the sample.

Assume we have example sequences x^k with residues x_j^k in column j of the alignment (we have adjusted our notation slightly; this x_j^k is not the jth residue in sequence x^k if there are gaps, but it is a convenient notation for what we need here). Once again, we need the conditional probabilities $P(b|a)$ derived from the substitution matrix. Let the residue in the common ancestor be y_j. Then we can use Bayes rule to calculate the posterior probability that $y_j = a$

$$P(y_j = a|\text{alignment}) = \frac{q_a \prod_k P(x_j^k|a)}{\sum_{a'} q_{a'} \prod_k P(x_j^k|a')}. \tag{5.5}$$

Note that we needed a prior distribution for residues at the common ancestor, which we set to q_a because that is our background probability for amino acids in the absence of further information.

We can now calculate the HMM emission probabilities as the predicted probabilities for a new sequence

$$e_{\mathrm{M}_j}(a) = \sum_{a'} P(a|a') P(y_j = a'|\text{alignment}). \tag{5.6}$$

One problem with this approach is that, as we noticed above, different columns vary widely in their degree of conservation. Indeed, that is one of the properties that we wanted to exploit when using alignments to estimate profile HMMs. However, using a single substitution matrix implies assuming a fixed degree of conservation. As we discussed in Chapter 2, matrices typically come in families varying in their level of implied conservation. Examples are the PAM [Dayhoff, Schwartz & Orcutt 1978] and the BLOSUM [Henikoff & Henikoff 1992] series of matrices. We can therefore significantly improve the approach in (5.5) and (5.6) if we optimise over choice of matrix from a family. This way, a very conserved column might use a conservative matrix, such as PAM30, and a very varied column would use a divergent matrix, such as PAM500.

How do we choose the optimal matrix? A natural approach is to maximise the likelihood of the observed data

$$P(x_j^1, \ldots, x_j^N | t) = \sum_a q_a \prod_k P(x_j^k | a, t) \tag{5.7}$$

where t is the matrix family parameter (t for evolutionary *time*). It would also be possible to use a Bayesian approach here, proposing a prior distribution over t, then combining this with (5.7) in Bayes' rule to obtain a posterior distribution for t, and summing over this in (5.6). However, that would require signficantly more computation.

The maximum likelihood time-dependent approach is closely related to the 'evolutionary weights' method in the PROFILE package [Gribskov & Veretnik 1996]. However, that method estimates different evolutionary times t for each possible ancestral amino acid, and also adjusts the resulting weights with respect to a set of baseline probabilities; for details see Gribskov & Veretnik [1996]. There are also strong connections between the methods of this subsection and those discussed later in Chapter 8 when building phylogenetic trees using maximum likelihood methods.

Testing the pseudocount methods

All the methods mentioned above have been tested in various ways. Direct tests, in which profiles were constructed and used for searching, were carried out extensively by Henikoff & Henikoff [1996]. The best method turned out to be the substitution matrix based method (5.6), with $A = 5R$ as described above; the Dirichlet mixture regulariser came a reasonably close second. Other tests gave different results [Tatusov, Altschul & Koonin 1994; Karplus 1995], so it is not clear which method is best, and it is likely that this will depend on the application and the details of the mixture components or substitution matrix used.

An interesting method was for testing various regularisers was given by Karplus [1995]. Instead of performing a huge number of database searches, he

asked the following question:[3] How well can an amino acid distribution be approximated from a small sample? Columns were extracted from a large set of deep alignments (the BLOCKS database; Henikoff & Henikoff [1991]). Imagine we take a small sample of size n with counts s_a from a column with complete counts C_a. From the sample counts s_a we can estimate the probabilities $e_s(a)$ of other symbols that might occur in the same column, using one of the methods described above (we use a subscript s to remind ourselves that this estimation is dependent on the sample counts). We can also estimate the probabilities of other symbols directly from the frequencies with which they occur in all columns of the database together with the probability $P(s|C)$ of drawing s from a column C (given by the multinomial distribution). This estimate is given by:

$$P(a|s) = \frac{\sum_{\text{columns } C} P(s|C)C_a}{\sum_{\text{columns } C} P(s|C)|C|},$$

where $|C|$ denotes the number of symbols in the column C. $P(a|s)$ can only be calculated up to a sample size of $n = 5$, but this is also the most interesting regime, because it is for small sample sizes that regularisation is most crucial. We can now use the relative entropy $-\sum_a P(a|s)\log e_s(a)$ to compare the 'ideal' probability $P(a|s)$ with that given by the regulariser. Summing over all samples s of size n gives a measure

$$E_n = \sum_{s,|s|=n} P(s)\left(-\sum_a P(a|s)\log e_s(a)\right), \tag{5.8}$$

where $P(s)$ is the probability of drawing the sample s averaged over all columns in the database. This can be calculated using $P(s) = \sum_C P(s|C)|C|/\sum_C |C|$.

Karplus proposed that a good regulariser should minimise E_n. He showed that several of the more complex regularisers described above resulted in estimators that were very close to optimal, in the sense that E_n was very small up to $n = 5$. Of course, we are ultimately interested in database searches, and it is not evident that the regulariser obtaining the lowest value of E_n will actually be best for searching. It is likely that the typical similarities in the source alignment database are not the same as the ones that we will be searching for with our HMM.

As well as evaluating methods, Karplus' approach can also be used to set the free parameters in the various methods described above, for example the total number of pseudocounts A to use in (5.3). For any value of A we can calculate E_n from our database of columns, either directly or by some sort of random sampling, and in fact we can also calculate the gradient of the relative entropy with respect to A. We can therefore find the value of A that minimises this average relative entropy, using gradient descent methods [Press *et al.* 1992], or by

[3] This page has been rewritten for the second printing.

other optimisation methods. In principle this can be done for any sample size n, yielding parameters dependent on n.

5.7 Optimal model construction

When we first discussed the parameterisation of profile HMMs, we pointed out that as well as estimating the probability parameters, it is necessary to decide which columns of the alignment should be assigned to insert states, and which to match states. We call this process model construction. At the time we proposed a simple heuristic, but we can do better than that. There is an efficient dynamic programming algorithm which can find the column assignments that maximise the posterior probability of the mode, at the same time as fitting optimal probability parameters.

In the profile HMM formalism, it is assumed that an aligned column of symbols corresponds either to emissions from the same match state or to emissions from the same insert state. It therefore suffices to mark which columns come from match states to specify a profile HMM architecture and the state paths for all the sequences in the alignment, as shown in Figure 5.7. In a marked column, symbols are assigned to match states and gaps are assigned to delete states. In an unmarked column, symbols are assigned to insert states and gaps are ignored. State transition and symbol emission counts are obtained from the state paths, and these counts can be used to estimate probability parameters by one of the methods in the previous section. In passing, we note that this model estimation procedure implicitly assumes that the multiple alignment is correct, i.e. that the implied state paths have probability one and all other state paths have probability zero, which is akin to a Viterbi assumption. The next chapter addresses issues of simultaneous alignment and model estimation.

There are 2^L combinations of markings for an alignment of L columns, and hence 2^L different profile HMMs to choose from. There are at least three ways to determine the marking. In *manual* construction, the user marks alignment columns by hand. This is perhaps the simplest way to allow users to manually specify the model architecture to use for a given alignment. In *heuristic* construction, a rule is used to decide whether a column should be marked. For instance, a column might be marked when the proportion of gap symbols in it is below a certain threshold. In *MAP* construction, a maximum *a posteriori* choice is determined by dynamic programming. A description of this algorithm follows.

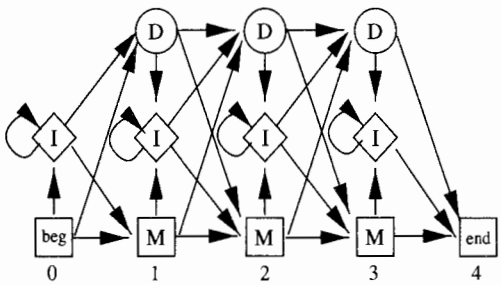
MAP match–insert assignment

The MAP construction algorithm recursively calculates a number S_j, which is the log probability of the optimal model for the alignment up to and including column

(a) Multiple alignment:

```
          x x . . . x
bat       A G - - - C
rat       A - A G - C
cat       A G - A A -
gnat      - - A A A C
goat      A G - - - C
          1 2 . . . 3
```

(b) Profile-HMM architecture:



(c) Observed emission/transition counts

		model position 0	1	2	3
match emissions	A	-	4	0	0
	C	-	0	0	4
	G	-	0	3	0
	T	-	0	0	0
insert emissions	A	0	0	6	0
	C	0	0	0	0
	G	0	0	1	0
	T	0	0	0	0
state transitions	M-M	4	3	2	4
	M-D	1	1	0	0
	M-I	0	0	1	0
	I-M	0	0	2	0
	I-D	0	0	1	0
	I-I	0	0	4	0
	D-M	-	0	0	1
	D-D	-	1	0	0
	D-I	-	0	2	0

Figure 5.7 *As an example of model construction from an alignment, a small DNA multiple alignment is given (a), with three columns marked above with x's. These three columns are assigned to positions 1–3 in the model architecture (b). The assignment of columns to model positions determines the symbol emission and state transition counts (c) from which probability parameters would be estimated.*

j, assuming that column j is marked. S_j is calculated from smaller subalignments ending at a marked column i ($i < j$) by incrementing S_i with the summed log probability of the transitions and emissions for the columns between i and j. The relevant probability parameters are estimated 'on the fly' from the counts that are implied by marking columns i and j while leaving unmarked the intervening columns (if any).

Transition and emission counts for a section of alignment bounded by marked columns i and j are independent of how columns are marked before i and after j, thus making a dynamic programming recursion possible. Only marked columns are considered in the recursion, because transition and emission counts for unmarked columns are not independent of the assignment of neighbouring columns; a single insert state may account for more than one column in the alignment.

For instance, let $\mathcal{T}_{ij}$ be the summed log probability of all the state transitions between marked columns i and j. We can determine $\mathcal{T}_{ij}$ from the observed state transition counts c_{xy} and the probabilities t_{xy}:

$$\mathcal{T}_{ij} = \sum_{x,y \in \mathrm{M,D,I}} c_{xy} \log a_{xy}.$$

Transition counts c_{xy} are obtained from the partial state paths implied by marking i and j. For instance, if in one sequence we see a gap in column i, five residues in columns $i+1$ to $j-1$, and a residue in column j, we would count one delete–insert transition, four insert–insert transitions, and one insert–match transition. The transition probabilities a_{xy} are estimated from the c_{xy} in the usual fashion, possibly including Dirichlet prior terms α_{xy} (or indeed, any form of prior that is independent of the marking outside of $i,\ldots,j$):

$$a_{xy} = \frac{c_{xy} + \alpha_{xy}}{\sum_y c_{xy} + \alpha_{xy}}.$$

Let $\mathcal{M}_j$ be the analogous log probability contribution for match state symbol emissions in column j, and $\mathcal{I}_{i+1,j-1}$ be the same for the insert state emissions for columns $i+1,\ldots,j-1$ (for $j-i>1$). We can now give the algorithm:

Algorithm: MAP model construction

Initialisation:

$S_0 = 0$, $\mathcal{M}_{L+1} = 0$.

Recurrence: for $j = 1,\ldots,L+1$:

$$S_j = \max_{0\le i<j} S_i + \mathcal{T}_{ij} + \mathcal{M}_j + \mathcal{I}_{i+1,j-1} + \lambda;$$

$$\sigma_j = \operatorname*{argmax}_{0\le i<j} S_i + \mathcal{T}_{ij} + \mathcal{M}_j + \mathcal{I}_{i+1,j-1} + \lambda.$$

Traceback: From $j = \sigma_{L+1}$, while $j > 0$:

Mark column j as a match column;

$j = \sigma_j$. ◁

A profile HMM is then built from the marked alignment. The extra term λ is a penalty used to favour models with fewer match states. In Bayesian terms, λ is the log of the prior probability of marking each column, implying a simple but adequate exponentially decreasing prior distribution over model lengths.

With some care in implementation, this algorithm is $O(L)$ in memory and $O(L^2)$ in time for an alignment of L columns.

5.8 Weighting training sequences

One issue that we have avoided completely so far is that of weighting sequences when estimating parameters. In a typical alignment, there are often some sequences that are very closely related to each other. Intuitively, some of the information from these sequences is shared, so we should not give them each the same influence in the estimation process as a single sequence that is more highly diverged from all the others. In the extreme that two sequences are identical, it makes sense that they should each get half the weight of other sequences, so that

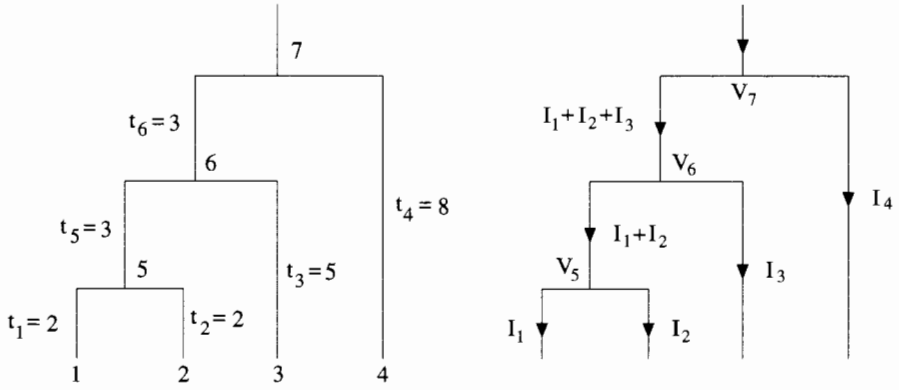


Figure 5.8 *On the left, a tree of sequences with branch lengths. On the right, the corresponding 'current' and 'voltage' values used in the 'Kirchhoff's law' approach to sequence weighting (see text).*

the net effect is of having only one of them. Statistically, the problem is that typically the examples we have do not constitute a good random sample from all the sequences that belong to the family; the assumption of independence is incorrect. To deal with this sort of situation, there have been a large number of proposals for different ways to assign weights to sequences. In principle, any of these can be used in combination with any of the methods of the preceding sections on fitting model parameters and model construction.

Simple weighting schemes derived from a tree

Many weighting approaches are based on building a tree relating the sequences. Since sequences in a family are related by an evolutionary tree, a very natural approach is to try to reconstruct this tree and use it when estimating the independent contribution of each of the observed sequences, downweighting sequences that have only recently diverged. We discuss phylogenetic tree construction at length later in Chapters 7 and 8, as well as in the next chapter on multiple sequence alignment. For our current purposes, the fine details of the method are probably not too important, and we will assume that we are given a tree connecting the sequences, with branch lengths indicating the relative degrees of divergence for each edge in the tree.

One of the intuitively simplest weighting schemes [Thompson, Higgins & Gibson 1994b] can be expressed nicely as follows. We are given a tree made of a conducting wire of constant thickness and apply a voltage V to the root. All the leaves are set to zero potential and the currents flowing from them are measured and taken to be the weights. Clearly, the currents will be smaller in the highly divided parts of the tree so these weights have the right qualitative properties. They

can be calculated by applying Kirchhoff's laws. For instance, in the tree shown in Figure 5.8, let the current and voltage at node n be I_n and V_n, respectively. Since constant factors do not affect the calculation, we can set the resistance equal to the edge-time. We then find $V_5 = 2I_1 = 2I_2$, $V_6 = 2I_1 + 3(I_1 + I_2) = 5I_3$, and $V_7 = 8I_4 = 5I_3 + 3(I_1 + I_2 + I_3)$. There are therefore three equations relating the four currents, and these give $I_1 : I_2 : I_3 : I_4 = 20 : 20 : 32 : 47$.

Another attractively simple idea was proposed by Gerstein, Sonnhammer & Chothia [1994]. Their algorithm works up the tree from the leaves, incrementing the weights. Initially the weight of a sequence is set equal to the edge-time of the edge immediately above it. Now, suppose node n has been reached. The edge above n has edge-time t_n, and this is shared out amongst the weights of all the sequences at the leaves below n, incrementing them by a fraction proportional to their current weight values. Formally, the increase Δw_i in a weight w_i is given by

$$\Delta w_i = t_n \frac{w_i}{\sum_{\text{leaves } k \text{ below } n} w_k}. \tag{5.9}$$

The same operation is carried out up to the root.

This is clearly an easy and efficient algorithm. For instance, the weights in the tree of Figure 5.8 are computed as follows: Initially the weights are set to the edge lengths of the leafs, $w_1 = w_2 = 2$, $w_3 = 5$, and $w_4 = 8$. At node 5 the edge length of 3 above node 5 is shared out equally to w_1 and w_2, giving them 3/2 each, so now $w_1 = w_2 = 2 + 3/2 = 3.5$. At node 6 we find the edge of length 3 above node 6 is shared out to nodes 1, 2 and 3 in the ratio 3.5 : 3.5 : 5, making $w_1 = w_2 = 3.5 + 3 \times 3.5/12$, and $w_3 = 5 + 3 \times 5/12$. With $w_4 = 8$, this gives $w_1 : w_2 : w_3 : w_4 = 35 : 35 : 50 : 64$. Even though these weights are close to those given by the Kirchhoff rule, the methods are in a sense opposed, for in a tree with two leaves and one edge longer than the other, the longer edge is down weighted by Kirchhoff and up weighted by (5.9).

Root weights from Gaussian parameters

One view of weights is that they should represent the influence of leaves on the root distribution. It is possible to make this idea precise, as Altschul, Carroll & Lipman [1989] showed. They built on the version of Felesenstein's 'pruning' algorithm which applies to continuous parameters [Felsenstein 1973]. Instead of discrete members of an alphabet we have a continuous real-valued variable, like the weight of an organism. In place of a substitution matrix we have a probability density that defines the probability of substituting one value, x, of this variable by another, y. A simple example of such a density is a Gaussian, where the probability of $x \to y$ along an edge with time t is $\exp(-(x-y)^2/(2\sigma^2 t^t)$. The

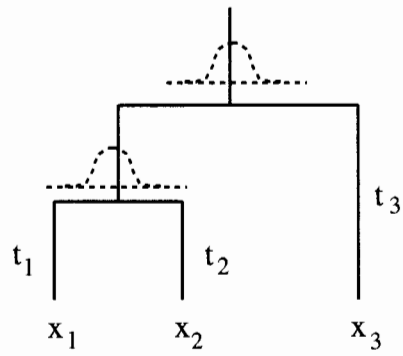


Figure 5.9 *The tree described in the text when deriving Gaussian weights.*

pruning algorithm now proceeds exactly as for a finite alphabet, but with integrals replacing discrete sums [Felsenstein 1973].[3]

Felsenstein's algorithm yields a Gaussian distribution for the parameter in question at the root whose mean μ depends linearly on the values x_i of the parameters at the leaves, so $\mu = \sum w_i x_i$. Altschul, Carroll & Lipman [1989] proposed that these w_i should be used as weights. They represent the influence of each leaf at the root.

Example: Altschul–Carroll–Lipman weights for a three-leaf tree

To illustrate how the weights are derived, consider the simple three-leaf tree shown in Figure 5.9, where leaf i takes the value x_i. The probability distribution at node 4 is given by

$$P(x \text{ at node } 4 \mid L_1, L_2) = K_1 \mathrm{e}^{-\frac{(x-x_1)^2}{2t_1}} \mathrm{e}^{-\frac{(x-x_2)^2}{2t_2}}$$

where K_1 is a normalising constant. One can rewrite this as

$$P(x \text{ at node } 4 \mid L_1, L_2) = K_1 \mathrm{e}^{-\frac{(x-v_1x_1-v_2x_2)^2}{2t_{12}}}$$

where $v_1 = t_2/(t_1+t_2)$, $v_2 = t_1/(t_1+t_2)$ and $t_{12} = t_1t_2/(t_1+t_2)$. If we were considering only the two-leaf tree with root at node 4, the mean of the root distribution would be given by $\mu = v_1x_1 + v_2x_2$, and the weights would be v_1 and v_2. Continuing with our three-leaf tree, however, we find next that the distribution at node 5

[3] Historically, the continuous case came first, and Felsenstein defined the pruning algorithm for Gaussian distributions of real-valued parameters. In the cited paper he takes account of the distribution of the parameters at each leaf, e.g. the mean and variance of the weight of an organism. Puzzlingly, he also introduces covariances between values for different leaves. It is not clear how to calculate a covariance between, say, the weights of cows and cats. For proteins, having multiple corresponding sites in an alignment would allow correlations to be considered in principle.

is given by

$$P(y \text{ at node } 5 \mid L_1, L_2, L_3) = K_2 \mathrm{e}^{-\frac{(y-x_3)^2}{2t_3}} \int \mathrm{e}^{-\frac{(x-v_1x_1-v_2x_2)^2}{2t_{12}}} \mathrm{e}^{-\frac{(x-y)^2}{2t_4}} dx$$

where K_2 is a normalising constant, and the integral is taken over all possible values of x at node 4 (and is the exact equivalent of the sum over all possible ancestral assignments of residues in the case of a discrete alphabet). This is a standard Gaussian integral, and boils down to the following

$$P(y \text{ at node } 5 \mid L_1, L_2, L_3) = K_3 \mathrm{e}^{-\frac{(y-w_1x_1-w_2x_2-w_3x_3)^2}{2t_{123}}}$$

where K_3 is a new normalising constant and $t_{123} = t_3\{t_1t_2 + t_4(t_1 + t_2)\}/\Omega$, with $\Omega = t_1t_2 + (t_3 + t_4)(t_1 + t_2)$. The mean of the distribution of y, i.e. of the root distribution, is given by

$$\mu = w_1x_1 + w_2x_2 + w_3x_3$$

with $w_1 = t_2t_3/\Omega$, $w_2 = t_1t_3/\Omega$, and $w_3 = \{t_1t_2 + t_4(t_1 + t_2)\}/\Omega$. These are therefore the Altschul–Carroll–Lipman weights for a tree with three leaves. □

Voronoi weights

There are also weighting schemes not based on trees. One approach is based on an image of the sequences from a family lying in 'sequence space'. In general, some will lie in clusters and others will be widely separated. The philosophy of the Voronoi scheme [Sibbald & Argos 1990] is to assume that this unevenness represents effects of sampling, including the 'sampling' performed by natural selection in favouring certain phyla. A more thorough trawl through all eligible sequences of the protein family, or perhaps a multitude of reruns of evolution, should produce a flat distribution within some region. To compensate for the gaps, we want to give sequences a weight proportional to the volume of empty space around them.

If sequence space were two-dimensional, or even low-dimensional, we could use standard methods from computational geometry to divide up space into regions around each example point. The standard approach is to take lines joining neighbouring pairs of points and draw their perpendicular bisectors, extending them till they join up. This produces a partitioning into polygons (in two dimensions) called a *Voronoi diagram* [Preparata & Shamos 1985], which has the property that the polygon around each point is the set of all points closer to that point than any other.

Sequence space is of course a high-dimensional construct in which the Voronoi geometry is hard to picture or calculate. However, we can implement the underlying principle of it by sampling sequences randomly from sequence space and testing to see which of the family sequences each sequence lies closest to. The

trick is in the sampling. This is accomplished by choosing, at each position of the alignment, uniformly from those residues which occur at that position in any sequence. If we count n_i such sample sequences closest to the ith family member (dividing up the counts if there is a tie), then we can define the ith weight to be $n_i / \sum_k n_k$.

Maximum discrimination weights

Another approach to weighting comes indirectly, from focusing initially on a reformulation of the primary goal in building the model [Eddy, Mitchison & Durbin 1995]. Rather than maximising the likelihood of sequences in the family, or even their posterior probability derived from Bayesian priors, we are normally interested in making the correct decision on whether sequences are members of the family or not. We are therefore interested in the probability

$$P(M|x) = \frac{P(x|M)P(M)}{P(x|M)P(M) + P(x|R)(1 - P(M))}.$$

where x is a sequence from the family, M is the model for the family that we are fitting, R is our alternative, random model for sequences not in the family, and $P(M)$ is the prior probability of a new sequence belonging to the family. Given example training sequences x^k, we would like to maximise the probability of classifying them all correctly, which is

$$D = \prod_k P(M|x^k),$$

not $\prod P(x^k|M)$ as usual with maximum likelihood based approaches. We call D the *discrimination* of the model on the set of sequences x^k. Maximising D will have the effect of emphasising performance on distant or difficult members of the family. Sequences that are easily classified will have $P(M|x)$ values very close to one; changing parameters to increase their likelihood $P(x|M)$ will have very little effect on D. On the other hand, increasing the likelihood of sequences for which $P(M|x)$ is small can potentially have a big effect.

It turns out that the parameter values that maximise D can be shown to be the ones that maximise a weighted version of the likelihood, where the weights are proportional to $1 - P(M|x_i)$, i.e. the probability of misclassifying sequence i. This can be seen from the observation that if $y = e^x/(K + e^x)$, then

$$\frac{\partial \log y}{\partial x} = \frac{1}{K + e^x} = K(1 - y).$$

If we set $x = \log\left(\frac{P(x|M)}{P(x|R)}\right)$, which is the log likelihood ratio for sequence x, then $y = P(M|x)$. So at a maximum of $\log D$ we will also be at a maximum of the weighted sum of log likelihood ratios, with weights $1 - P(M|x_i)$, and since the

random model is fixed this is equivalent to a maximum of the weighted log likelihood of the model M. The maximum discrimination criterion therefore amounts to another sequence weighting system.

One difference from previous systems, however, is that these weights are defined in a somewhat circular fashion; they depend upon the model that is being fit. When using maximum discrimination weighting as a method, an iterative approach must be used; an initial set of weights gives rise to a model, from which posterior probabilities $P(M|x)$ can be calculated, giving rise to new weights, and hence a new model, and so on until convergence is achieved. This iterative re-estimation procedure is analogous to the versions of the EM algorithm used to fit HMM parameters to sets of unlabelled sequences (p. 64 and p. 323).

Maximum discrimination training has a big advantage in that it is directly optimising performance on the type of operation that the model will be used for, ensuring that the most effort is applied to recognising the most distant sequences. On the other hand, exactly the same point can lead to problems. If there is any training sequence that has been misclassified, then the distortion needed to give it a good score can damage performance for correct members of the class. To some extent, though, this same problem occurs with all weighting schemes: incorrectly assigned sequences will be the most distant ones in any tree that gets built from the examples.

Maximum entropy weights

Finally, we describe two weighting methods based on the idea of trying to make the statistical spread of the model as broad as possible.

Assume column i of a multiple alignment has k_{ia} residues of type a and a total of m_i different types of residues. To make a distribution as uniform as possible from these counts by weighting each sequence, we can choose a weight for sequence k of $1/(m_i k_{ix_i^k})$. Maximum likelihood estimation will then yield a distribution $p_{ia} = k_{ia}/(m_i k_{ia}) = 1/m_i$, i.e. all the residues appearing in the column will have the same probability. To illustrate the idea, suppose we have ten sequences with residue A at a site, and one sequence with a B, so the unweighted frequencies of A and B are $c_{\mathrm{A}} = \frac{10}{11}$, $c_{\mathrm{B}} = \frac{1}{11}$. The weights of the ten sequences are $w_1 = w_2 = \ldots = w_{10} = 1/(2 \times 10) = 0.05$, and $w_{11} = 1/(2 \times 1) = 0.5$, which have the effect of making the overall weighting for each of A and B equal.

The preceding paragraph only considered one column. With just one weight per sequence, it is of course not possible to make the distribution uniform for all columns in an alignment. However, by averaging over all columns, one may hope to obtain reasonable weights. That is, the weights are calculated as

$$w_k = \sum_i \frac{1}{m_i k_{ix_i^k}},$$

and then normalised to sum to one. This weighting scheme was proposed by [Henikoff & Henikoff 1994].

Instead of averaging, there is another approach to combining the information from the different columns that has a simple theoretical justification. A standard measure of the 'uniformity' of a distribution is the entropy (11.8), which is larger the more uniform the distribution is. Indeed, it is easy to see that the weights chosen above based on a single column maximise the entropy of the distribution p_{ia} for that column. An HMM defines a probability distribution over sequences, and therefore a natural extension of the single column weighting to full sequences is to maximise the entropy of the complete HMM distribution [Krogh & Mitchison 1995]. We will see that, perhaps surprisingly, this is closely related to maximum discrimination weighting.

Let us consider all the sites in an alignment with no gaps. We then sum the entropies from each site, and choose the weights to maximise this sum: that is we maximise $\sum_i H_i(w_\bullet) + \lambda \sum_k w_k$, where $H_i(w_\bullet) = \sum_a p_{ia} \log p_{ia}$, and p_{ia} is the weighted frequency of residue a at the ith site, computed as above.

Suppose for instance that we have the sequences $x^1 = \mathrm{AEA}$, $x^2 = \mathrm{AAC}$, and $x^3 = \mathrm{DAC}$. Giving them weights w_1, w_2 and w_3, respectively, the entropies at each site are

$$\begin{aligned} H_1(w_\bullet) &= -(w_1+w_2)\log(w_1+w_2) - w_3 \log w_3, \\ H_2(w_\bullet) &= -w_1 \log w_1 - (w_2+w_3)\log(w_2+w_3), \\ H_3(w_\bullet) &= -w_1 \log w_1 - (w_2+w_3)\log(w_2+w_3). \end{aligned}$$

We assume that the weights sum to one, and therefore we have to use a Lagrange multiplier term $\lambda \sum_k w_k$, when differentiating and finding the maximum of the entropy. Setting the derivatives of $H_1(w_\bullet) + H_2(w_\bullet) + H_3(w_\bullet) + \lambda \sum_k w_k$ to zero gives $(w_1+w_2)w_1^2 = (w_1+w_2)(w_2+w_3)^2 = w_3(w_2+w_3)^2$, which implies $w_1 = w_3 = 0.5, w_2 = 0$. This makes the frequencies in each column equal, which was our goal. If it seems odd to give a sequence zero weight, note that the residue at each site in x^2 is always present in one of the other two sequences. Intuitively, x^2 lies 'between' x^1 and x^3, (in fact, it would be a possible ancestral sequence of x^1 and x^3 in an evolutionary reconstruction based on parsimony; see Chapter 7).

Another way to view to the result of this example is that if we set the model probabilities to be the weighted counts frequencies, as a weighted maximum likelihood procedure would, the resulting model assigns an equal probability to all of the original sequences, x^1, x^2 and x^3. This seems very reasonable, according to the view that all the example sequences should be treated as equally good members of the family for which we are building the model. In fact, Krogh & Mitchison [1995] show that the maximum entropy procedure assigns weights to the example sequences so that some subset of the sequences (perhaps all of them) have non-zero weight and equal probabilities under the resulting model, or they

have a higher probability, in which case they have zero weight. The former can be thought of as boundary points for the region of sequence space occupied by the whole sequence set, while the latter are internal points.

Furthermore, empirical tests indicate that the maximum entropy weights are optimal in the sense that they maximise the minimum score assigned to any of the example sequences [Krogh & Mitchison 1995]. This is an absolute version of the criterion specified in the previous section on maximum discrimination weights; rather than simply weighting the weakest match most strongly, all the parameter-fitting effort is applied to increasing its score, until it reaches that of the other non-zero-weighted sequences. Although satisfying an attractive goal, maximum entropy weighting suffers from the same problems as maximum discrimination: if a sequence is an outlier that should not be a full member of the family, the method will force it in, possibly at a substantial cost in performance on all other sequences. In addition, the rejection of all information from some of the sequences may seem intuitively undesirable.

Exercise

5.6 Compute the weights for the following sequence set, using each of the weighting methods described above except Voronoi weights (which requires random sampling of sequences): AGAA, CCTC, AGTC.

5.9 Further reading

PSSM methods were introduced during the 1980s for finding new members of sequence families, although the matrix values were not always obtained using an explicit probability-based derivation. They are also known by other names, such as *weight matrices* [Staden 1988]. More recent papers using related methods include those by Stormo [1990]; Henikoff & Henikoff [1994]; Tatusov, Altschul & Koonin [1994].

The non-probabilistic versions of profiles already have a long history, and many variants of the profile method have been suggested and tested. Thompson, Higgins & Gibson [1994b] and Luthy, Xenarios & Bucher [1994] report an improvement when the sequences are weighted using one of the BLOSUM matrices [Henikoff & Henikoff 1992] instead of a PAM matrix. In Thompson, Higgins & Gibson [1994b] the treatment of gaps is also improved.

Several ways have been suggested for incorporating structural information into profiles. In Luthy, McLachlan & Eisenberg [1991] substitution matrices were estimated for six different structural environments: the three secondary structure elements α-helix, β-sheet, and 'other' combined with an outside/inside classification, which was based on the exposure of an amino acid to solvent. Other vari-