# Chapter 4

# Methods Applied in Immunological Bioinformatics

A large variety of methods are commonly used in the field of immunological bioinformatics. In this chapter many of these techniques are introduced. The first section describes the powerful techniques of weight-matrix construction, including sequence weighting and pseudocount correction. The techniques are introduced using an example of peptide-MHC binding. In the following sections the more advanced methods of Gibbs sampling, ANNs, and hidden Markov models (HMMs) are introduced. The chapter concludes with a section on performance measures for predictive systems and a short section introducing the concepts of representative data set generation.

## 4.1 Simple Motifs, Motifs and Matrices

In this section, we shall demonstrate how simple but reasonably accurate prediction methods can be derived from a set of training data of very limited size. The examples selected relate to peptide-MHC binding prediction, but could equally well have been related to proteasomal cleavage, TAP binding, or any other problem characterized by simple sequence motifs.

A collection of sequences known to contain a given binding motif can be used to construct a simple, data-driven prediction algorithm. Table 4.1 shows a set of peptide sequences known to bind to the HLA-A*0201 allele.

From the set of data shown in table 4.1, one can construct simple rules defining which peptides will bind to the given HLA molecule with high affinity. From the above example it could, e.g., be concluded that a binding motif must

| |
|---|
| ALAKAAAAM |
| ALAKAAAAN |
| ALAKAAAAV |
| ALAKAAAAT |
| ALAKAAAAV |
| GMNERPILT |
| GILGFVFTM |
| TLNAWVKVV |
| KLNEPVLLL |
| AVVPFIVSV |

Table 4.1: Small set of sequences of peptides known to bind to the HLA-A*0201 molecule.

be of the form

$$X_1[LMIV]_2X_3X_4X_5X_6X_7X_8[MNTV]_9\,, \tag{4.1}$$

where $X_i$ indicates that all amino acids are allowed at position $i$, and $[LMIV]_2$ indicates that only the specified amino acids L, M, I, and V are allow at position 2. Following this approach, two peptides with $T$ and $V$ at position 9, respectively, will be equally likely to bind. Since $V$ is found more often than $T$ at position 9, one might, however, expect that the latter peptide is more likely to bind. We will later discuss in more detail why positions 2 and 9 are of special importance.

Using a statistical approach, such differences can be included directly in the predictions. Based on a set of sequences, a probability matrix $p_{pa}$ can be constructed, where $p_{pa}$ is the probability of finding amino acid $a$ ($a$ can be any of the 20 amino acids) on position $p$ ($p$ can be 1 to 9 in this example) in the motif. In the above example $p_{9V} = 0.4$ and $p_{9T} = 0.2$. This can be viewed as a statistical model of the binding site. In this model, it is assumed that there are no correlations between the different positions, e.g., that the amino acid present on position 2 does not influence which amino acids are likely to be observed on other positions among binding peptides.

The probability [also called the likelihood p(sequence|model)] of observing a given amino acid sequence $a_1a_2\ldots a_p\ldots$ given the model can be calculated by multiplying the probabilities for observing amino acid $a_1$ on position 1, $a_2$ on position 2, etc. This product can be written as

$$\prod_p p_{pa}\,. \tag{4.2}$$

Any given amino acid sequence $a_1a_2\ldots a_p\ldots$ may also be observed in a randomly chosen protein. Furthermore, long sequences will be less likely than

short ones. The probability p(sequence|background model) of observing the sequence in a random protein, can be written as

$$\prod_p q_a, \tag{4.3}$$

where $q_a$ is the background frequency of amino acid $a$ on position $p$. The index $p$ has been left out on $q_a$ since it is normally taken to be equal on all positions.

The ratio of these two likelihoods is called the odds ratio $O$,

$$O = \frac{\prod_p p_{pa}}{\prod_p q_a} = \prod_p \frac{p_{pa}}{q_a} . \tag{4.4}$$

The background amino acid frequencies $q_a$ define a so-called null model. Different null models can be used: the amino acid distribution in a large set of proteins such as the Swiss-Prot database [Bairoch and Apweiler, 2000], a flat distribution (all amino acid frequencies $q_a$ are set to 1/20), or an amino acid distribution estimated from sequences known not to be binders (negative examples). If the odds ratio is greater than 1, the sequence is more likely given the model than given the background model.

The odds ratio can be used to predict if a peptide is likely to bind. Multiplying many probabilities may, however, result in a very low number that in computers are rounded off to zero (numerical underflow). To avoid this, prediction algorithms normally use logarithms of odds ratios called log-odds ratios.

The score $S$ of a peptide to a motif is thus normally calculated as the sum of the log-odds ratio

$$S = \log_k \left( \prod_p \frac{p_{pa}}{q_a} \right) = \sum_p \log_k \left( \frac{p_{pa}}{q_a} \right) , \tag{4.5}$$

where $p_{pa}$ as above is the probability of finding amino acid $a$ at position $p$ in the motif, $q_a$ is the background frequency of amino acid $a$, and $\log_k$ is the logarithm with base $k$. The scores are often normalized to half bits by multiplying all scores by $2/\log_k(2)$. The logarithm with base 2 of a number $x$ can be calculated using a logarithm with another base $n$ (such as the natural logarithm with base $n = e$ or the logarithm with base $n = 10$) using the simple formula $\log_2(x) = \log_n(x)/\log_n(2)$. In half-bit units, the log-odds score $S$ is then given as

$$S = 2 \sum_p \log_2 \left( \frac{p_{pa}}{q_a} \right) . \tag{4.6}$$

## 4.2 Information Carried by Immunogenic Sequences

Once the binding motif has been described by a probability matrix $p_{pa}$, a number of different calculations can be carried out characterizing the motif.

### 4.2.1 Entropy

The entropy of a random variable is a measure of the uncertainty of the random variable; it is a measure of the amount of information required to describe the random variable [Cover and Thomas, 1991]. The entropy $H$ (also called the Shannon entropy) of an amino acid distribution $p$ is defined as

$$H(p) = -\sum_a p_a \log_2(p_a) \,, \tag{4.7}$$

where $p_a$ is the probability of amino acid $a$. Here the logarithm used has the base of 2 and the unit of the entropy then becomes bits [Shannon, 1948]. The entropy attains its maximal value $\log_2(20) \simeq 4.3$ if all amino acids are equally probable, and becomes zero if only one amino acid is observed at a given position. We here use the definition that $0\log(0) = 0$. For the data shown in table 4.1 the entropy at position 2 is, e.g., found to be $\simeq 1.36$.

### 4.2.2 Relative Entropy

The relative entropy can be seen as a distance between two probability distributions, and is used to measure how different an amino acid distribution $p$ is from some background distribution $q$. The relative entropy is also called the Kullback-Leibler distance $D$ and is defined as

$$D(p\|q) = \sum_a p_a \log_2(\frac{p_a}{q_a}) \,. \tag{4.8}$$

The background distribution is often taken as the distribution of amino acids in proteins in a large database of sequences. Alternatively, $q$ and $p$ can be the distributions of amino acids among sites that are known to have or not have some property. This property could, e.g., be glycosylation, phosphorylation, or MHC binding.

The relative entropy attains its maximal value if only the least probable amino acid according to the background distribution is observed. The relative entropy is non-negative and becomes zero only if $p = q$. It is not a true metric, however, since it is not symmetric ($D(p\|q) \neq D(q\|p)$) and does not satisfy the triangle inequality ($D(p\|q) \not< D(p\|r) + D(r\|q)$) [Cover and Thomas, 1991].

### 4.2.3 Logo Visualization of Relative Entropy

To visualize the characteristics of binding motifs, the so-called sequence logo technique [Schneider and Stephens, 1990] is often used. The information content at each position in the sequence motif is indicated using the height of a column of letters, representing amino acids or nucleotides. For proteins the information content is normally defined as the relative entropy between the amino acid distribution in the motif, and a background distribution where all amino acids are equally probable. This gives the following relation for the information content:

$$I = \sum_a p_a \log_2 \frac{p_a}{1/20} = \log_2(20) + \sum_a p_a \log_2 p_a \ . \tag{4.9}$$

The information content is a measure of the degree of conservation and has a value between zero (no conservation; all amino acids are equally probable) and $\log_2(20) \simeq 4.3$ (full conservation; only a single amino acid is observed at that position). In the logo plot, the height of each letter within a column is proportional to the frequency $p_a$ of the corresponding amino acid $a$ at that position. When another background distribution is used, the logos are normally called Kullback-Leibler logos, and letters that are less frequent than the background are displayed upside down.

In logo plots, the amino acids are normally colored according to their properties:

- Acidic [DE]: red
- Basic [HKR]: blue
- Hydrophobic [ACFILMPVW]: black
- Neutral [GNQSTY]: green

But other color schemes can be used if relevant in a given context. An example of a logo can be seen in Figure 4.1.

### 4.2.4 Mutual Information

Another important quantity used for characterizing a motif is the mutual information. This quantity is a measure of correlations between different positions in a motif. The mutual information measure is in general defined as the reduction of the uncertainty due to another random variable and is thus a measure of the amount of information one variable contains about another. Mutual information between two variables is defined as

$$I(A;B) = \sum_a \sum_b p_{ab} \log_2 \left(\frac{p_{ab}}{p_a p_b}\right) \ , \tag{4.10}$$

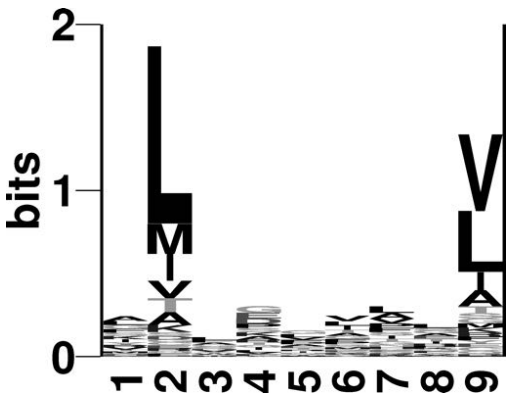



Figure 4.1: Logo showing the bias for peptides binding to the HLA-A*0201 molecule. Positions 2 and 9 have high information content. These are anchor positions that to a high degree determine the binding of a peptide [Rammensee et al., 1999]. See plate 4 for color version.

where $p_{ab}$ is the joint probability mass function (the probability of having amino acid $a$ in the first distribution and amino acid $b$ in the second distribution) and

$$p_a = \sum_b p_{ab} \ , \ p_b = \sum_a p_{ab} \ . \tag{4.11}$$

It can be shown that [Cover and Thomas, 1991],

$$I(A;B) = H(A) - H(A|B) \tag{4.12}$$

where $H$ is the entropy defined in equation(4.7). From this relation, we see that uncorrelated variables have zero mutual information since $H(A|B) = H(A)$ for such variables. The mutual information attains its maximum value, $H(A)$, when the two variables are fully correlated, since $H(A|B) = 0$ in this case. The mutual information is always non-negative. Mutual information can be used to quantify the correlation between different positions in a protein, or in a peptide-binding motif. Mutations in one position in a protein may, e.g., affect which amino acids are found at spatially close positions in the folded protein. Mutual information can be visualized as matrix plots [Gorodkin et al., 1999]. Figure 4.2 gives an example of a mutual information matrix plot for peptides binding to MHC alleles within the A2 supertype. For an explanation of supertypes, see chapter 13.

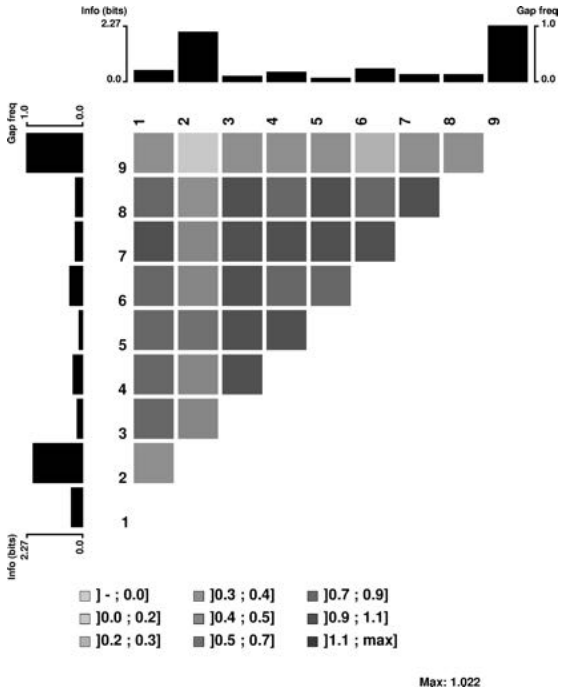



Figure 4.2: Mutual information plot calculated from peptides binding to MHC alleles within the A2 supertype. The plot was made using MatrixPlot [Gorodkin et al., 1999] (http://www.cbs.dtu.dk/services/MatrixPlot/).

## 4.3 Sequence Weighting Methods

In the following, we will use the logo plots to visualize some problems one often faces when deriving a binding motif characterized by a probability matrix $p_{pa}$ as described in section 4.1.

The values of $p_{pa}$ may be set to the frequencies $f_{ab}$ observed in the alignment. There are, however, some problems with this direct approach. In figure 4.3, a logo representation of the probability matrix calculated from the peptides in table 4.1 is shown. From the plot, it is clear that alanine has a very high probability at all positions in the binding motif. The first 5 sequences in the alignment are very similar, and may reflect a sampling bias, rather than an actual amino acids bias in the binding motif. In such a situation, one would therefore like to downweight identical or almost identical sequences.

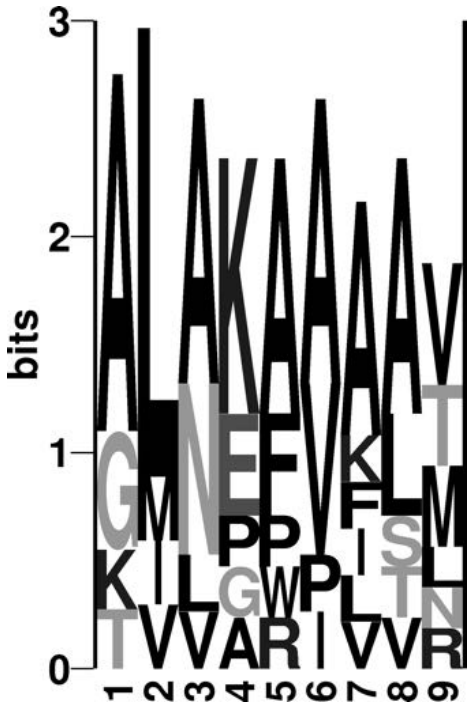



Figure 4.3: Logo representation of the probability matrix calculated from 10 9mer peptides known to bind HLA-A*0201.

Different methods can be used to weight sequences. One method is to cluster sequences using a so-called Hobohm algorithm [Hobohm et al., 1992]. The Hobohm algorithm (version 1) takes an ordered list of sequences as input. From the top of the list sequences are placed on an accepted list or discarded depending on whether they are similar (share more than $X$% identify to any member on the accepted list) or not. This procedure is repeated for all sequences in the list. After the Hobohm reduction, the pairwise similarity in the accept list therefore has a maximum given by the threshold used to generate it.

This method is also used for the construction of the BLOSUM matrices normally used by BLAST. The most commonly used clustering threshold is 62%. After the clustering, each peptide $k$ in a cluster is assigned a weight $w_k = 1/N_c$, where $N_c$ is the number of sequences in the cluster that contains peptide $k$. When the amino acid frequencies are calculated, each amino acid in

sequence $k$ is weighted by $w_k$. In the above example the first 5 peptides will form one cluster, and each of these sequences thus contributes with a weight of $\frac{1}{5}$ to the probability matrix. The frequency of $A$ at position $p_1$ will then be $p_{1A} = 2/6 = 0.33$ as opposed to $6/10 = 0.6$ found when using the raw sequence counts.

In the Henikoff and Henikoff [1994] sequence weighting scheme, an amino acid $a$ on position $p$ in sequence $k$ contributes a weight $w_{kp} = 1/rs$, where $r$ is the number of *different* amino acids at a given position (column) in the alignment and $s$ the number of occurrences of amino acid $a$ in that column. The weight of a sequence is then assigned as the sum of the weights over all positions in the alignment. The Henikoffs' method is fast as the computation time only increases linearly with the number of sequences. For the Hobohm clustering algorithm, on the other hand, computation time increases as the square of the number of sequences (depending on the similarity between the sequences). Performing the sequence weighting using clustering generally leads to more accurate results, and clustering is the suggested choice of method if the number of sequences is limited and the calculation thus computationally feasible.

Figure 4.4 shows a logo representation of the probability matrix calculated using clustering sequence weighting. From the figure it is apparent that the strong alanine bias in the motif has been removed.

## 4.4 Pseudocount Correction Methods

Another problem with the direct approach to estimating the probability matrix $p_{pa}$ is that the statistics often will be based on very few sequence examples (in this case 10 sequences). A direct calculation of the probability $p_{9I}$ for observing an isoleucine on position 9 in the alignment, e.g., gives 0. This will in turn mean that all peptides with an isoleucine on position 9 will score minus infinity in equation (4.5), i.e., be predicted not to bind no matter what the rest of the sequence is. This may be too drastic a conclusion based on only 10 sequences. One solution to this problem is to use a pseudocount method, where prior knowledge about the frequency of different amino acids in proteins is used. Two strategies for pseudocount correction will be described here: Equal and BLOSUM correction, respectively. In both cases the pseudocount frequency $g_{pa}$ for amino acid $a$ on position $p$ in the alignment is estimated as described by Altschul et al. [1997],

$$g_{pa} = \sum_b \frac{f_{pb}}{q_b}\, q_{ab} = \sum_b f_{pb}\, q_{a|b}\ . \tag{4.13}$$

Here, $f_{pb}$ is the observed frequency of amino acid $b$ on position $p$, $q_b$ is the background frequency of amino acid $b$, $q_{ab}$ is the frequency by which amino


Figure 4.4: Logo representation of the probability matrix calculated from 10 9mer peptides known to bind HLA-A*0201. The probabilities are calculated using the clustering sequence weighting method.

acid $a$ is aligned to amino acid $b$ derived from the BLOSUM substitution matrix, and $q_{a|b}$ is the corresponding conditional probability. The equation shows how the pseudo-count frequency can be calculated. The pseudocount frequency for isoleucine at position 9 in the example in table 4.1 would, e.g., be

$$g_{9I} = \sum_b f_{9b}\, q_{I|b} = 0.3\, q_{I|V} + 0.2\, q_{I|T} \ldots 0.1\, q_{I|L} \simeq 0.09\,, \tag{4.14}$$

where here, for simplicity, we have used the raw count values for $f_{9b}$. In real applications the sequence-weighted probabilities are normally used. The $q_{a|b}$ values are taken from the BLOSUM62 substitution matrix [Henikoff and Henikoff, 1992].

In the Equal correction, a substitution matrix with identical frequencies for all amino acids (1/20) and all amino acid substitutions (1/400) is applied. In this case $g_{pa} = 1/20$ at all positions for all amino acids.

## 4.5 Weight on Pseudocount Correction

From estimated pseudocounts, and sequence-weighted observed frequencies, the effective amino acid frequency can be calculated as [Altschul et al., 1997]

$$p_{pa} = \frac{\alpha f_{pa} + \beta g_{pa}}{\alpha + \beta} \,. \tag{4.15}$$

Here $f_{pa}$ is the observed frequency (calculated using sequence weighting), $g_{pa}$ the pseudocount frequency, $\alpha$ the effective sequence number minus 1, and $\beta$ the weight on the pseudocount correction. When the sequence weighting is performed using clustering, the effective sequence number is equal to the number of clusters. When sequence weighting as described by Henikoff and Henikoff [1992] is applied, the average number of different amino acids in the alignment gives the effective sequence number. If a large number of different sequences are available $\alpha$ will in general also be large and a relative low weight will thus be put on the pseudocount frequencies. If, on the other hand, the number of observed sequences is one, $\alpha$ is zero, and the effective amino acid frequency is reduced to the pseudocount frequency $g_{pa}$. If we calculate the log-odds score $S$, for a $G$, as given by equation (4.5), $G$ gets the score:

$$S_G = \log \frac{g_{pG}}{q_G} = \log \frac{q_{GG}}{q_G q_G} \,, \tag{4.16}$$

where we have used equation (4.13) for $g_{pa}$. The last log-odds score is the BLOSUM matrix score for $G - G$, and we thus find that the log-odds score for a single sequence reduces to the BLOSUM identical match score values.

Figure 4.5 shows the logo plot of the probability matrix calculated from the sequences in table 4.1, including sequence weighting and pseudocount correction. The figure demonstrates how the pseudocount correction allows for probability estimates for all 20 amino acids at all positions in the motif. Note that $I$ is the fifth most probable amino acid at position 9, even though this amino acid was never observed at the position in the peptide sequences.

## 4.6 Position Specific Weighting

In many situations prior knowledge about the importance of the different positions in the binding motif exists. Such prior knowledge can with success be included in the search for binding motifs [Lundegaard et al., 2004, Rammensee et al., 1997]. In figure 4.6, we show the results of such a position-specific weighting. The figure displays the probability matrix calculated from the 10 sequences and a matrix calculated from a large set of 485 peptides. It demonstrates how a reasonably accurate motif description can be derived from a very

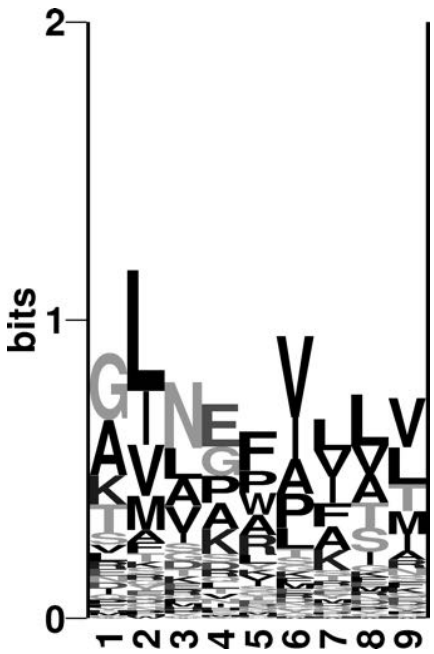



Figure 4.5: Logo representation of the probability matrix calculated from 10 9mer peptides known to bind HLA-A*0201. The probabilities are calculated using both the methods of sequence weighting and pseudocount correction.

limited set of data, using the techniques of sequence weighting, pseudocount correction, and position-specific weighting.

## 4.7 Gibbs Sampling

In previous sections, we have described how a weight matrix describing a sequence motif can be calculated from a set of peptides of equal length. This approach is appropriate when dealing with MHC class I binding, where the length of the binding peptides are relatively uniform. MHC class II molecules, on the other hand, can bind peptides of very different length, and the weight-matrix methods described up to now are hence not directly applicable to characterize this type of motif. Here we describe a motif sampler suited to deal with such problems.

The general problem to be solved by the motif sampler is to locate and

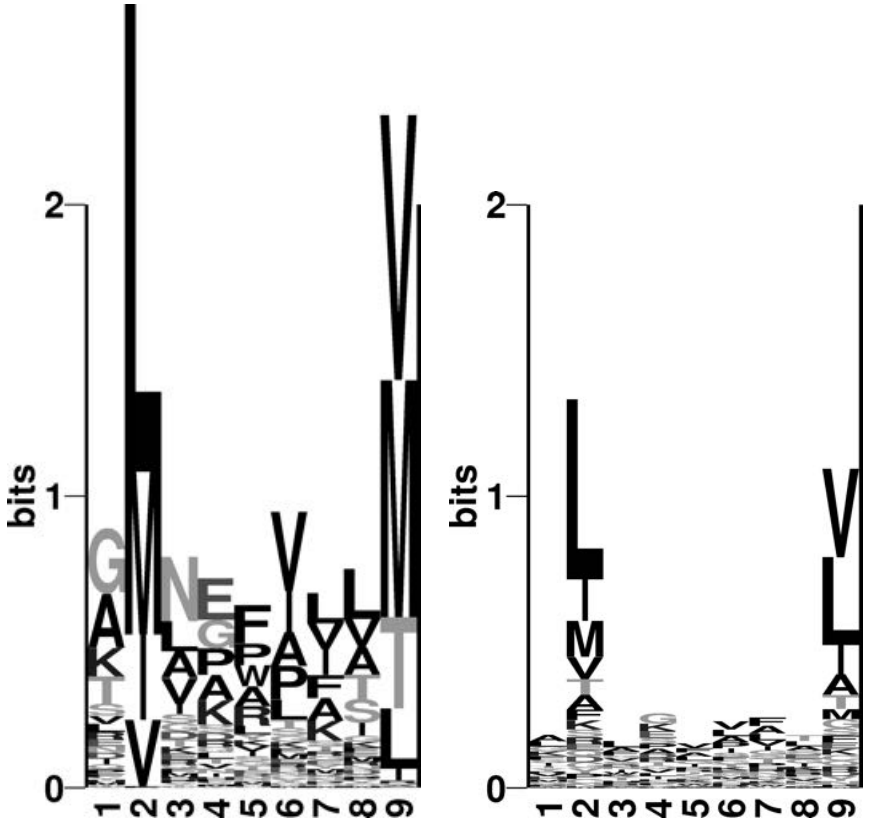



Figure 4.6: Left: Logo representation of the probability matrix calculated from 10 9mer peptides known to bind HLA-A*0201. The probabilities are calculated using the methods of sequence weighting, pseudocount correction, and position-specific weighting. The weight on positions 2 and 9 is 3. Right: Logo representation of the probability matrix calculated from 485 peptides known to bind HLA-A*0201.

characterize a pattern embedded within a set of $N$ amino acids (or DNA) sequences. In situations where the sequence pattern is very subtle and the motif weak, this is a highly complex task, and conventional multiple sequence alignment programs will typically fail. The Gibbs sampling method was first described by Lawrence et al. [1993] and has been used extensively for location of transcription factor binding sites [Thompson et al., 2003] and in the analysis of protein sequences [Lawrence et al., 1993, Neuwald et al., 1995]. The method attempts to find an optimal local alignment of a set of N sequences

by means of Metropolis Monte Carlo sampling [Metropolis et al., 1953] of the alignment space. The scoringfunction guiding the Monte Carlo search is defined in terms of fitness (information content) of a log-odds matrix calculated from the alignment.

The algorithm samples possible alignments of the $N$ sequences. For each alignment a log-odds weight matrix is calculated as $\log(p_{pa}/q_a)$, where $p_{pa}$ is the frequency of amino acid $a$ at position $p$ in the alignment and $q_a$ is the background frequency of that amino acid. The values of $p_{pa}$ can be estimated using sequence weighting and pseudocount correction for low counts as described earlier in this chapter.

The fitness (energy) of an alignment is calculated as

$$E = \sum_{p,a} C_{pa} \log \frac{p_{pa}}{q_a} , \tag{4.17}$$

where $C_{pa}$ is the number of times amino acid $a$ is observed at position $p$ in the alignment, $p_{pa}$ is the pseudocount and sequence weight corrected amino acid frequency of amino acid $b$ and position $p$ in the alignment. Finally, $q_a$ is the background frequency of amino acid $a$. $E$ is equal to the sum of the relative entropy or the Kullback-Leibler distance [Kullback and Leibler, 1951] in the window.

The set of possible alignments is, even for a small data set, very large. For a set of 50 peptides of length 10, the number of different alignments with a core window of nine amino acids is $2^{50} \simeq 10^{15}$. This number is clearly too large to allow for a sampling of the complete alignment space. Instead, the Metropolis Monte Carlo algorithm is applied [Metropolis et al., 1953] to perform an effective sampling of the alignment space.

Two distinct Monte Carlo moves are implemented in the algorithm: (1) the single sequence move, and (2) the phase shift move. In the single sequence move, the alignment of a sequence is shifted a randomly selected number of positions. In the phase shift move, the window in the alignment is shifted a randomly selected number of residues to the left or right. This latter type of move allows the program to efficiently escape local minima. This may, e.g., occur if the window overlaps the most informative motif, but is not centered on the most informative pattern.

The probability of accepting a move in the Monte Carlo sampling is defined as

$$P = \min(1, e^{dE/T}) , \tag{4.18}$$

where $dE$ is difference in (fitness) energy between the end and start configurations and $T$ is a scalar. Note that we seek to maximize the energy function, hence the positive sign for $dE$ in the equation. $T$ is a scalar that is lowered during the calculation. The equation implies that moves that increase $E$ will

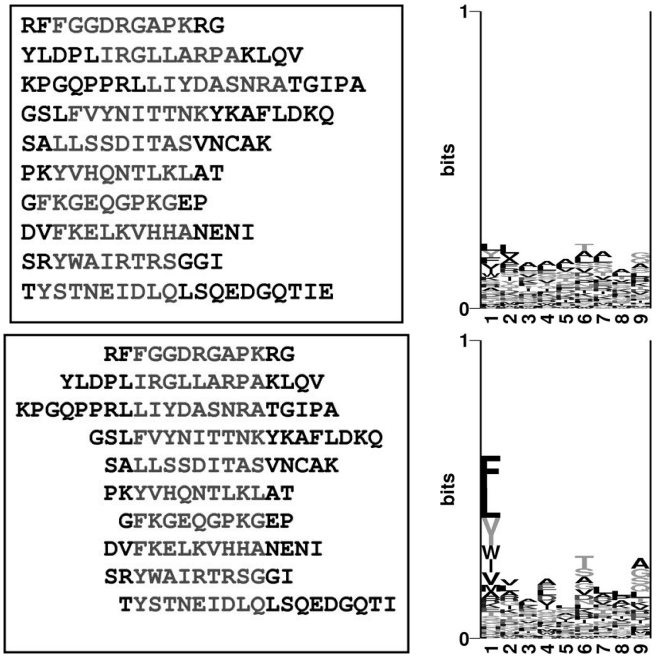



Figure 4.7: Example of an alignment generated by the Gibbs sampler for the DR4(B1*0401) binding motif. The peptides were downloaded from the MHCPEP database [Brusic et al., 1998a]. Top left: Unaligned sequences. Top right: Logo for unaligned sequences. Bottom left: Sequences aligned by Gibbs sampler. Bottom right: Logo for sequences aligned by the Gibbs sampler. Reprinted, with permission, from Nielsen et al. [2004]. See plate 5 for color version.

always be accepted ($dE > 0$). On the other hand, only a fraction given by $e^{dE/T}$ of the moves which decrease E will be accepted. For high values of the scalar $T$ ($T \gg dE$) this probability is close to 1, but as T is lowered during the calculation, the probability of accepting unfavorable moves will be reduced, forcing the system into a state of high fitness (energy). Figure 4.7 shows a set of sequences aligned by their N-terminal (top left) and the corresponding logo (top right). The lower panel shows the alignment by the Gibbs sampler and the corresponding logo. The figure shows how the Gibbs sampler has identified a motif describing the binding to the DR4(B1*0401) allele. For more details on the Gibbs sampler see Chapter 8.

## 4.8 Hidden Markov Models

The Gibbs sampler and other weight-matrix approaches are well suited to describe sequence motifs of fixed length. For MHC class II, the peptide binding motif is in most situations assumed to be of a fixed length of 9 amino acids. This implies that the scoringfunction for a peptide binding to the MHC complex can be written as a linear sum of 9 terms. In many situations this simple motif description is, however, not valid. In the previous chapter, we described how protein families, e.g, often are characterized by conserved amino acid regions separated by amino acid segments of variable length. In such situations a weight matrix approach is poorly suited to characterize the motif. HMMs, on the other hand, provide a natural framework for describing such interrupted motifs.

In this section, we will give a brief introduction to the HMM framework. First, we describe the general concepts of the HMM framework through a simple example. Next the Viterbi and posterior decoding algorithms for aligning a sequence to a HMM are explained, and finally the use of HMMs in some selected biological problems is described. A detailed introduction to HMMs and their application to sequence analysis problems may be found, e.g., in Durbin et al. [1998] and Baldi and Brunak [2001].

### 4.8.1 Markov Model, Markov Chain

A Markov model consists of a set of states. Each state is associated with a probability distribution assigning probability values to the set of possible outcomes. A set of transition probabilities for switching between the states is assigned. In a Markov model (or Markov chain) the outcome of an event depends only on the preceding state.

An example of such a model is a B cell epitope model. Regions in the sequence with many hydrophobic residues are less likely to be exposed on the surface of proteins and it is therefore less likely that antibodies can bind to these regions. In this model, we divide positions in a protein in two states: epitopes $E$ and non-epitopes $N$. We divide the 20 different amino acids in three groups. Hydrophobic [ACFILMPVW] , uncharged polar [GNQSTY] and charged [DEHKR]. This model is displayed in Figure 4.8. Even though this model is highly simplified and does only capture the most simple, of the very complex, features describing the B cell epitopes, it serves the purpose of introducing the important concepts of an HMM.

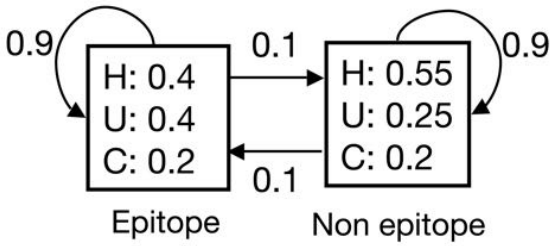



Figure 4.8: B cell epitope model. The model has two states: Epitope $E$ and non epitope $N$. In each state, three different types of amino acids can be found Hydrophobic (H), uncharged polar (U) and charged (C). The transition probabilities between the two states are given next to the arrows, and the probability of each of the three types of amino acids are given for each of the two states.

### 4.8.2 What is Hidden?

What is hidden in the HMM? In biology HMMs are most often used to assign a state (epitope or non-epitope in this example) to each residue in a biological sequence (3 types of amino acids in this example). An HMM can, however, also be used to construct artificial sequences based on the probabilities in it. When the model is used in this way, the outcome (often called the emissions) is a sequence like $HHHUHHCH\ldots$. It is not possible from the observed sequence to establish if the model for each letter was in the epitope state or not. This information is kept hidden by the model.

### 4.8.3 The Viterbi Algorithm

Even though the list of states used by the HMM to generate the observed sequence is hidden, it is possible to obtain an accurate estimate of the list of states used. If we have an HMM like the one described in figure 4.8, we can use a dynamic programming algorithm like the one described in chapter 3 to align the observed sequence to the model and obtain the path (list of states) that most probably will generate the observations. The dynamic programming algorithm doing the alignment of a sequence to the HMM is called the Viterbi algorithm.

If the highest probability $P_k(x_i)$ of a path ending in state $k$ with observation $x_i$ is known for all states $k$, then the highest probability for observation $x_{i+1}$ in state $l$, can be found as

$$P_l(x_{i+1}) = p_l(x_{i+1}) \max_k(P_k(x_i)a_{kl}) \ , \tag{4.19}$$

where $p_l(x_{i+1})$ is the probability of observation $x_{i+1}$ in state $l$, and $a_{kl}$ is the transition probability from state $k$ to state $l$.

By using this relation recursively, one can find the path through the model that most probably will give the observed sequence. To avoid underflow in the computer the algorithm normally will work in log-space and calculate $\log P_l(x_{i+1})$ instead. In log-space the recursive equation becomes a sum, and the numbers remain within a reasonable range.

An example of how the Viterbi algorithm is applied is given in figure 4.9. The figure shows how the optimal path through the HMM of figure 4.8 is calculated for a sequence of $NGSLFWIA$. By translating the sequence into the three states defining hydrophobic, neutral and charged residues, we get $HHHUUUUU$. In the example, we assume that the model is the non-epitope state at the first $H$, which implies that is $P_E(H_1) = -\infty$. The value for assigning $H$ to the state $N$ is $P_N(H_1) = \log(0.55) = -0.26$. For the next residue, the path must come from the $N$ state. We therefore find, $P_N(H_2) = \log(0.55) + \log(0.9) - 0.26 = -0.57$, and $P_E(H_2) = \log(0.4) + \log(0.1) - 0.26 = -1.66$, since $a_{NN}0.9$, and $a_{NE} = 0.1$. The backtracking arrows are for both the $E$ and the $N$ state placed to the previous $N$ state. For the third residue the path to the $N$ state can come from both the $N$ and the $E$ states. The value $P_N(H_3)$ is therefore found using the relation

$$P_N(H_3) = \log(0.55) + \max\{\log(0.9) - 0.57, \log(0.1) - 1.66\} = -0.88 \quad (4.20)$$

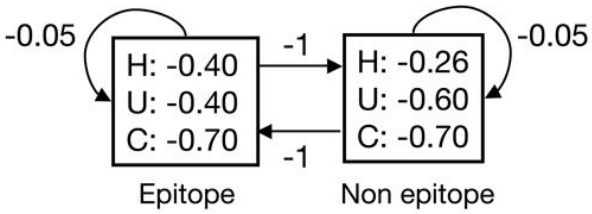
and likewise the value $P_E(H_3)$ is

$$P_E(H_3) = \log(0.4) + \max\{\log(0.1) - 0.57, \log(0.9) - 1.66\} = -1.97 \quad (4.21)$$

In both cases the max function selects the first argument, and the backtracking arrows are therefore for both the $E$ and the $N$ state assigned to the previous $N$ state. This procedure is repeated for all residues in the sequence, and we obtain the result shown in Figure 4.9. With the arrows, it is indicated which state was selected in the $\max_k$ function in each step in the recursive calculation. Repeating the calculation for all residues in the observed sequence, we find that the highest score $-4.08$ is found in state $E$. Backtracking through the arrows, we find the optimal path to be $EEENNNNN$ (indicated with solid arrows). Note that the most probable path of the sequence $HHHUUUU$ would have ended in the state $N$ with a value of $-3.48$, and the corresponding path would hence have been $NNNNNNN$. Observing a series of uncharged amino acids thus does not necessarily mean that the epitope state was used.

### 4.8.4 The Forward-Backward Algorithm and Posterior Decoding

Many different paths through an HMM can give rise to the same observed sequence. Where the Viterbi algorithm gives the most probable path through an




| | N | G | S | L | F | W | I | A |
|---|---|---|---|---|---|---|---|---|
| | H | H | H | N | N | N | N | N |
| E | NULL | -1.66 | -1.97 | **-2.28** | **-2.73** | **-3.18** | **-3.63** | **-4.08** |
| N | **-0.26** | **-0.55** | **-0.88** | -1.53 | -2.18 | -2.85 | -3.48 | -4.13 |
| | N | N | N | E | E | E | E | E |

Figure 4.9: Alignment of sequence *HHHUUUUU* to the B cell epitope model of figure 4.8. The upper part of the figure shows the log-transformed HMM. The probabilities have been transformed by taking the logarithm with base 10. The model is assumed to start in the non-epitope state at the first $H$. The table in the lower part gives the $\log P_l(x_{i+1})$ values for the different observations in the N (non epitope), and E (epitope) states, respectively. The arrows show the backtracking pointers. The solid arrows give the optimal path, the dotted arrows denote the suboptimal path. The upper two rows in the table give the amino acid and three letter transformed sequence, respectively . The lower row gives the most probable path found using the Viterbi algorithm.

HMM given the observed sequence, the so-called forward algorithm calculates the probability of the observed sequence being aligned to the HMM. This is done by summing over all possible paths generating the observed sequence. The forward algorithm is a dynamic programming algorithm with a recursive formula very similar to the Viterbi equation, replacing the maximization step with a sum [Durbin et al., 1998]. If $f_k(x_{i-1})$ is the probability of observing the sequence up to and including $x_{i-1}$ ending in state $k$, then the probability of observing the sequence up to and including $x_i$ ending in state $l$ can be found using the recursive formula

$$f_l(x_i) = p_l(x_i) \sum_k f_k(x_{i-1}) a_{kl} \,. \tag{4.22}$$

Here $p_l(x_i)$ is the probability of observation $x_i$ in state $l$, and $a_{kl}$ is the transition probability from state $k$ to state $l$.

Another important algorithm is the posterior decoding or forward-backward algorithm. The algorithm calculates the probability that an observation $x_i$ is aligned to the state $k$ given the observed sequence $x$. The term "posterior decoding" refers to the fact that the decoding is done *after* the sequence is observed. This probability can formally be written as $P(\pi_i = k|x)$ and can be determined using the so-called forward-backward algorithm [Durbin et al., 1998].

$$P(\pi_i = k|x) = \frac{f_k(i)b_k(i)}{P(x)} \,. \tag{4.23}$$

The term $f_k(i)$ is calculated using the forward recursive formula from before,

$$f_k(i) = p_k(x_i) \sum_l f_l(x_{i-1}) a_{lk} \,, \tag{4.24}$$

and $b_k(i)$ is calculated using a backward recursive formula,

$$b_k(x_i) = \sum_l a_{kl} p_l(x_{i+1}) b_l(i+1) \,. \tag{4.25}$$

From these relations, we see why the algorithm is called forward-backward. $f_k(i)$ is the probability of aligning the sequence up to and including $x_i$ with a path ending in state $k$, and $b_k(i)$ is the probability of aligning the sequence $x_{i+1} \ldots x_N$ to the HMM starting from state $k$. Finally $P(x)$ is the probability of aligning the observed sequence to the HMM.

One of the most important applications of the forward-backward algorithm is the posterior decoding. Often many paths through the HMM will have probabilities very close to the optimal path found by the Viterbi algorithm. In such situations posterior decoding might be a more adequate algorithm to extract properties of the observed sequence from the model. Posterior decoding gives a list of states that most probably generate the observed sequence using the equation

$$\pi_i^{posterior} = \max_k P(\pi_i = k|x) \,, \tag{4.26}$$

where $P(\pi_i = k|x)$ is the probability of observation $x_i$ being aligned to state $\pi_k$ given the observed sequence $x$. Note that posterior decoding is different from the Viterbi decoding since the list of states found by posterior decoding need not be a legitimate path through the HMM.

### 4.8.5 Higher Order Hidden Markov Models

The central property of the Markov chains described until now is the fact that the probability of an observation only depends on the previous state and that

the probability of an observed sequence, $X$, thus can be written as

$$P(X) = P(x_1)P(x_2|x_1)P(x_3|x_2)\cdots P(x_N|x_{N-1}) \tag{4.27}$$

where $P(x_i)$ denotes the probability of observing $x$ at position $i$.

In many situations, this approximation might not be valid since the probability of an observation might depend on more than just the preceding state. However by use of higher order Markov models, such dependences can be captured. In a Markov model of n'th order, the probability of an observation $x_i$ is given by

$$P(x_i) = P(x_i|x_{i-1},\ldots,x_{i-n}) \tag{4.28}$$

A second order hidden Markov model describing B cell epitopes may thus consist of two states each with 9 possible observations $HH$, $HU$, $HC$, $UH$, $UU$, $UC$, $CH$, $CU$, and $CC$. By assigning different probability values to for instance the observations $HU$, $UU$ and $CU$, the model can capture higher order correlations.

An n'th order Markov model over some alphabet is thus equivalent to a first order Markov chain over an alphabet of n-tuples.

### 4.8.6 Hidden Markov Models in Immunology

Having introduced the HMM framework through a simple example, we now turn to some relevant biological problems that are well described using HMMs. The first is highly relevant to antigen processing, and describes how an HMM can be designed to characterize the binding of peptides to the human transporter associated with antigen processing (TAP). The second example addresses a more general use of HMMs in characterizing similarities between protein sequences, the so-called profile HMMs.

**TAP** Transport of the peptides into the endoplasmic reticulum is an essential step in the MHC class I presentation pathway. This task is done by TAP molecules and a detailed description of the function of the TAP molecules is given in chapter 7. The peptides binding to TAP have a rather broad length distribution, and peptides up to a length of 18 amino acids can be translocated [van Endert et al., 1994]. The binding of a peptide to the TAP molecules is to a high degree determined by the first three N-terminal positions and the last C-terminal position in the peptide. Other positions in the peptide determine the binding to a lesser degree. The binding of a peptide to the TAP molecules is thus an example of a problem where the binding motif has variable length, and hence a problem that is well described by a HMM. Figure 4.10 shows an HMM describing peptide TAP binding. The figure highlights the important differences and similarities between a weight matrix and an HMM. If we only

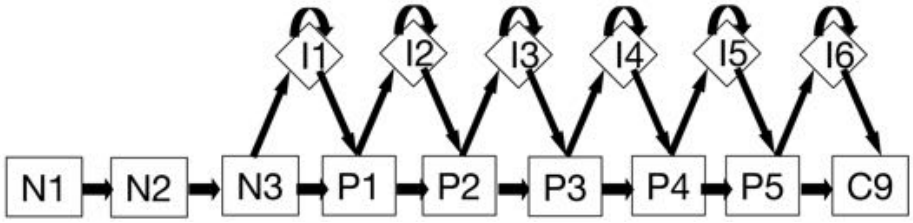



Figure 4.10: HMM for peptide TAP binding. The model can describe binding of peptides of different lengths to the TAP molecules. The binding motif consists of 9 amino acids. The first three N-terminal amino acids, and the last C-terminal amino acids must be part of the binding motif. Each state is associated with a probability distribution of matching one of the 20 amino acids. The arrow between the states indicates the transition probabilities for switching between the states. The amino acid probability distributions for each state are estimated using the techniques of sequence weighting and pseudocount correction (see section 4.4).

consider alignment of 9mer peptides to the HMM, we see that no alignment can go through the insertion states (labeled as I in the figure). In this situation the alignment becomes a simple sum of the amino acid match scores from each of the 9 states N1-N3, P1-P5, and C9, and the HMM is reduced to a simple weight matrix. However, if the peptide is longer than nine amino acids, the path through the HMM must pass some insertion state, and it is clear that such a motif could not have been characterized well by a weight matrix.

**Profile Hidden Markov Models** Profile HMMs are used to characterize sequence similarities within a family of proteins. As described in chapter 3 a multiple alignment of protein sequences within a protein family can reveal important information about amino acids conservation, mutability, active sites, etc.

A profile HMM provides a natural framework for compiling such information of a multiple alignment. In figure 4.11, we show an example of a profile HMM. The architecture of a profile HMM is very similar to the model for peptide TAP binding. The model is build from a set of match states (P1-P7). These states describe what is conserved among most sequences in the protein family. Some sequences within a family will have amino acid insertions; others will have amino acid deletions with respect to the motif. To allow for such variation in sequence, the profile HMM has insertion and deletion states (labeled as I and D in the figure, respectively). The model can insert amino acids between match states using the insertion state, and a match state can be skipped using the deletion states.

An example of a multiple alignment was given in figure 3.12C. From this type of alignment, one can construct a profile HMM. If we consider positions

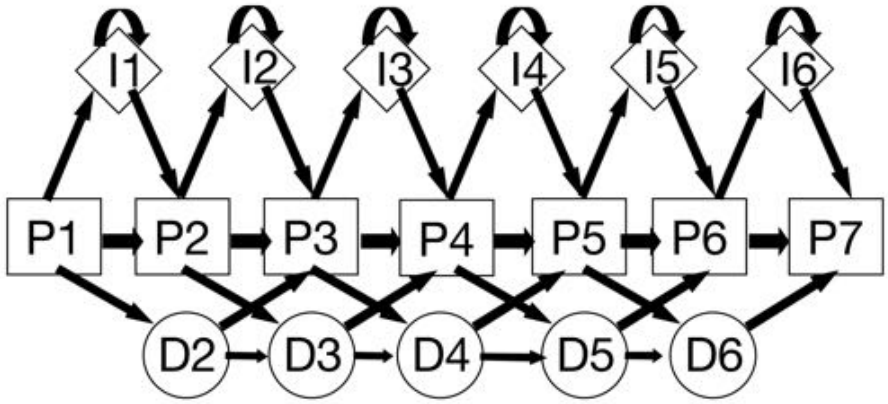



Figure 4.11: Profile HMM with 7 match states. Match states are shown as squares, insertion state as diamonds, and deletion states as circles. Each match and insertion state has an associated probability distribution for matching the 20 different amino acids. Transitions between the different states are indicated by arrows.

in the alignment with less than 40% gaps to be match states, then all other positions are either insertions or deletions. In the example in figure 3.12 *Neurospora crassa* and *Saccharomyces cerevisiae* hence contain an insertion in position 58-64, whereas positions 32-38 in *Saccharomyces cerevisiae*, and positions 35-38 in *Neurospora crassa* are deleted. Note that we count the positions in the alignment, not the positions in the sequence. The figure demonstrates that insertions and deletions are distributed in a highly nonuniform manner in the alignment. Also, it is apparent from the figure that not all positions are equally conserved. The W in position 72 is thus fully conserved in all species, whereas the W in position 53 is more variable. These variations in sequence conservation and in the probabilities for insertions and deletions are naturally described by an HMM, and profile HMMs have indeed been applied successfully to the identification of new and remote homolog members of families with well-characterized protein domains [Sonnhammer et al., 1997, Karplus et al., 1998, Durbin et al., 1998].

## 4.9 Artificial Neural Networks

As stated earlier the weight-matrix approach is only suitable for prediction of a binding event in situations where the binding specificity can be represented

independently at each position in the motif. In many (in fact most) situations this is not the case, and this assumption can only be considered to be an approximation. In the binding of a peptide to the MHC molecule the amino acids might, e.g., compete for the space available in the binding grove. The mutual information in the binding motif will allow for identification of such higher-order sequence correlations. An example of a mutual information calculation for peptides binding to the MHC class I complex is shown in figure 4.2.

Neural networks with a hidden layer are designed to describe sequence patterns with such higher-order correlations. Due to their ability to handle these correlations, hundreds of different applications within bioinformatics have been developed using this technique, and for that reason ANNs have been enjoying a renaissance, not only in biology but also in many other data domains.

Neural networks realize a method of computation that is vastly different from "rule-based techniques" with strict control over the steps in the calculation from data input to output. Conceptually, neural networks, on the other hand, use "influence" rather than control. A neural network consists of a large number of independent computational units that can influence but not control each other's computations. That such a system, which consists of a large number of unintelligent units, in their biological counterparts can be made to exhibit "intelligent" behavior is not directly obvious, but one can with some justification use the central nervous system in support of the idea. However, the ANNs obviously do not to any extent match the computing power and sophistication of biological neural systems.

ANNs are not programmed in the normal sense, but must be influenced by data — trained — to associate patterns with each other.

The neural network algorithm most often used in bioinformatics is similar to the network structure described by Rumelhart et al. [1991]. This network architecture is normally called a standard, feedforward multilayer perceptron. Other neural network architectures have also been used, but will not be described here. The most successful of the more complex networks involves different kinds of feedback, such that the network calculation on a given (often quite short) amino acid sequence segment possibly can depend on sequence patterns present elsewhere in the sequence. When analyzing nucleotide data the applications have typically been used also for long sequence segments, such as the determination of whether a given nucleotide belongs to a protein coding sequence or not. The network can in such a case be trained to take advantage of long-range correlations hundreds of nucleotide positions apart in a sequence.

The presentation of the neural network theory outlined below is based on the paper by Rumelhart et al. [1991], as well as the book by Hertz et al. [1991]. The training algorithm used to produce the final network is a steepest descent

method that learns a training set of input-output pairs by adjusting the network weight parameters such that the network for each input will produce a numerical value that is close to the desired target output (either representing disjunct categories, or real values such as peptide binding affinities). The idea with the network is to produce algorithms which can handle sequence correlations, and also classify data in a nonlinear manner, such that small changes in sequence input can produce large changes in output. The hope is that the network then will be able to reproduce what is well-known in biology, namely that many single amino acid substitutions can entirely disrupt a mechanism, e.g., by inhibiting binding.

The feedforward neural network consists of connected computing units. Each unit "observes" the other units' activity through its input connections. To each input connection, the unit attaches a weight, which is a real number that indicates how much influence the input in question is to have on that particular unit. The influence is calculated as the weight multiplied by the activity of the neuron delivering the input. The weight can be negative, so an input can have a negative influence. The neuron sums up all the influence it receives from the other neurons and thereby achieves a measure for the total influence it is subjected to. From this sum the neuron subtracts a threshold value, which will be omitted from the description below, since it can be viewed as a weight from an extra input unit, with a fixed input value of $-1$. The linear sum of the inputs is then transformed through a nonlinear, sigmoidal function to produce its output. The input layer units does not compute anything, but merely store the network inputs; the information processing in the network takes place in the internal, hidden layer (most often only one layer), and in the output layer. A schematic representation of this type of neural network is shown in figure 4.12.

### 4.9.1 Predicting Using Neural Networks: Conversion of Input to Output

Formally the calculation in a network with one hidden layer proceeds as follows. Let the indices $i$, $j$, and $k$ refer to the output, hidden, and input layers, respectively. The input neurons each receive an input $I_k$. The input to each of the hidden units is

$$h_j = \sum_k v_{jk} I_k, \tag{4.29}$$

where $v_{jk}$ is the weight on the input $k$ to the hidden unit $j$. The output from the hidden units is

$$H_j = g(h_j) \tag{4.30}$$

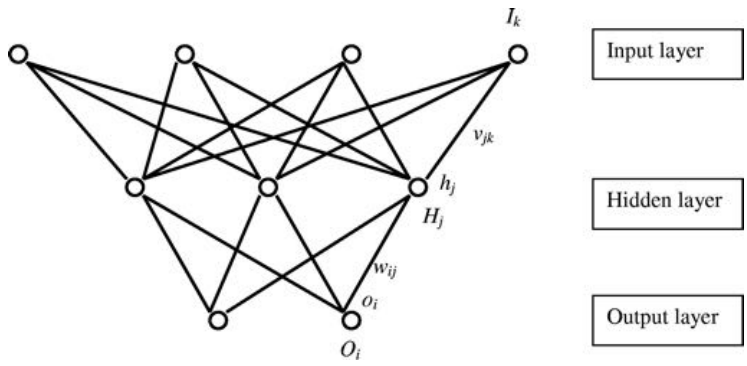



Figure 4.12: Schematic representation of a conventional feedforward neural network used in numerous applications within bioinformatics.

where

$$g(x) = \frac{1}{1 + e^{-x}} \tag{4.31}$$

is the sigmoidal function most often used. Note that

$$g'(x) = g(x)(1 - g(x)) \, . \tag{4.32}$$

Each output neuron receives the input

$$o_i = \sum_j w_{ij} H_j \, , \tag{4.33}$$

where $w_{ij}$ are the weights between the hidden and the output units to produce the final output

$$O_i = g(o_i) \, . \tag{4.34}$$

Different measures of the error between the network output and the desired target output can be used [Hertz et al., 1991, Bishop, 1995]. The most simple choice is to let the error $E$ be proportional to the sum of the squared difference between the desired output $d_i$ and the output $O_i$ from the last layer of neurons:

$$E = \frac{1}{2} \sum_i (O_i - d_i)^2 \, . \tag{4.35}$$

### 4.9.2 Training the Network by Backpropagation

One option is to update the weights by a back-propagation algorithm which is a steepest descent method, where each weight is changed in the opposite

direction of the gradient of the error,

$$\Delta w_{ij} = -\varepsilon \frac{\partial E}{\partial w_{ij}} \text{ and } \Delta v_{jk} = -\varepsilon \frac{\partial E}{\partial v_{jk}} . \tag{4.36}$$

The change of the weights between the hidden and the output layer can be calculated by using

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial O_i} \frac{\partial O_i}{\partial o_i} \frac{\partial o_i}{\partial w_{ij}} = \delta_i H_j , \tag{4.37}$$

where

$$\delta_i = (O_i - d_i) g'(o_i) . \tag{4.38}$$

To calculate the change of weights between the input and the hidden layer we use the following relations

$$\frac{\partial E}{\partial v_{jk}} = \frac{\partial E}{\partial H_j} \frac{\partial H_j}{\partial v_{jk}} , \tag{4.39}$$

and

$$\frac{\partial E}{\partial H_j} = \sum_i \frac{\partial E}{\partial o_i} \frac{\partial o_i}{\partial H_j} = \sum_i \frac{\partial E}{\partial o_i} w_{ij} , \tag{4.40}$$

and

$$\frac{\partial H_j}{\partial v_{jk}} = \frac{\partial H_j}{\partial h_j} \frac{\partial h_j}{\partial v_{jk}} = g'(h_j) I_k , \tag{4.41}$$

and thus

$$\frac{\partial E}{\partial v_{jk}} = g'(h_j) I_k \sum_i \delta_i w_{ij} . \tag{4.42}$$

In the equations described here the error is backpropagated after each presentation of a training example. This is called online learning. In batch, or offline, learning, the error is summed over all training examples and thereafter backpropagated. However, this method has proven inferior in most cases [Hertz et al., 1991].

In figure 4.13, we give a simple example of how the weights in the neural network are updated using backpropagation. The figure shows two configurations of a neural network with two hidden neurons. The network must be trained to learn the XOR (exclusive or) function. That is the function with the following properties:

$$\begin{aligned} f_{XOR}(0,0) &= f_{XOR}(1,1) = 0 \\ f_{XOR}(1,0) &= f_{XOR}(0,1) = 1 . \end{aligned} \tag{4.43}$$

This type of input-output association is the simplest example displaying higher-order correlation, as the two input properties are not independently

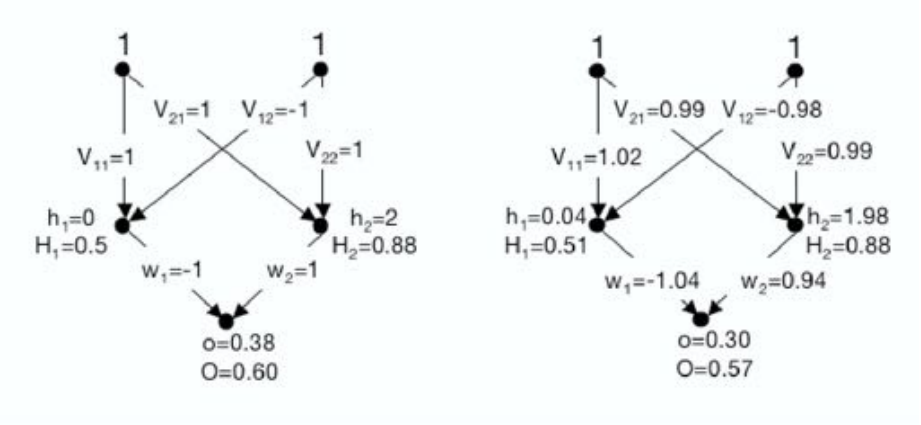



Figure 4.13: Update of weights in a neural network using backpropagation. The figure shows the neural network before updating the weights (left) and the network configuration after one round of backpropagation (right). The learning rate $\varepsilon$ in the example is equal to 0.5. Note that this is a large value for $\varepsilon$. Normally the value is of the order 0.05.

linked to the categories. The "1" category is represented by input examples where only one of the two features are allowed to be present — not both features simultaneously. The $(1,1)$ example from the "0" category is therefore an "exception," and this small data set can therefore not be handled by a linear network without hidden units. The example may seem very simple; still it captures the essence of the sequence properties in many binding sites, where the two features could be charge and side chain volume, respectively. In actual application the number of input features is typically much higher.

In the example shown in figure 4.13, we have for simplicity left out the threshold value normally subtracted from the input to each neuron. The figure shows the neural network before updating the weights and the network configuration after one round of backpropagation. With the example $(1,1)$, the network output, $O$, from the network with the initial weights is 0.6. This gives the following relation for $\delta$:

$$\delta = (0.6 - 0)g'(o) = 0.6 \cdot O \cdot (1 - O) = 0.15\,, \tag{4.44}$$

where we have used equation (4.32) for $g'(o)$.

The change of the weights from the hidden layer to the output neuron are updated using equation (4.37):

$$\Delta w_1 \quad = \quad -\varepsilon\, 0.15 \cdot 0.5 = -0.075\varepsilon$$

$$\Delta w_2 = -\varepsilon\, 0.15 \cdot 0.88 = -0.13\varepsilon\,. \tag{4.45}$$

The change of the weights in the first layer are updated using equation (4.42)

$$\begin{aligned}
\Delta v_{11} &= -\varepsilon\, g'(h_1) \cdot 1 \cdot \delta \cdot (-1) \\
&= \varepsilon\, H_1\, (1 - H_1) \cdot \delta \\
&= 0.04\varepsilon \\
\Delta v_{21} &= -\varepsilon\, g'(h_1) \cdot 1 \cdot \delta \cdot (-1) = 0.04\varepsilon \\
\Delta v_{12} &= -\varepsilon\, g'(h_2) \cdot 1 \cdot \delta \cdot 1 = -0.02\varepsilon \\
\Delta v_{22} &= -\varepsilon\, g'(h_2) \cdot 1 \cdot \delta \cdot 1 = -0.02\varepsilon\,.
\end{aligned} \tag{4.46}$$

Modifying the weights according to these values, we obtain the neural network configuration shown to the right of figure 4.13. The network output from the updated network is 0.57. Note that the error indeed has decreased. When the network is trained on all four patterns of the $XOR$ function during a number of training cycles (including the three threshold weights), the network will in most cases reach an optimal configuration, where the error on all four patterns is practically zero.

Figure 4.14 demonstrates how the XOR function is learned by the neural network. If we construct a neural network without a hidden layer this data set cannot be learned, whereas a network with two hidden neurons learns the four examples perfectly.

When examining the weight configuration of the fully trained network it becomes clear how the data set from the XOR function has been learned by the network. The XOR function can be written as

$$f_{XOR}(x_1, x_2) = (x_1 + x_2) - 2x_1x_2 = y - z\,, \tag{4.47}$$

where $y = x_1 + x_2$ and $z = 2x_1x_2$. From this relation, we see that the hidden layer allows the network to linearize the problem into a sum of two terms. The two functions $y$ and $z$ are encoded by the network using the properties of the sigmoid function. If we assume for simplicity that the sigmoid function is replaced by a step function that emits the value 1 if the input value is greater than or equal to the threshold value and zero otherwise, then the $y$ and $z$ functions can be encoded having the weights $v_{ij} = 1$ for all values of $i$ and $j$ and the corresponding threshold values 1 and 2 for the first and second hidden neuron, respectively. With these values for the weights and thresholds, the first hidden neuron will emit a value of 1 if either of the input values are 1, and zero otherwise. The second hidden neuron will emit a value of 1 only if both the input neurons are 1. Setting the weights $w_1 = 1$, and $w_2 = -1$, the network is now able to encode the XOR function.

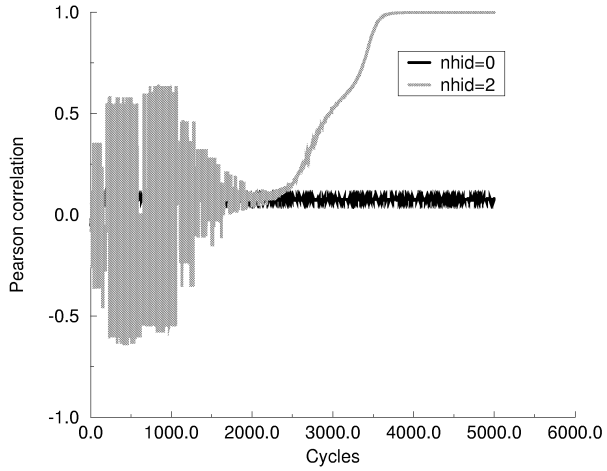



Figure 4.14: Neural network learning curves for nonlinear patterns. The plot shows the Pearson correlation as a function of the number of learning cycles during neural network training. The black curve shows the learning curve for the XOR function for a neural network without hidden neurons, and the gray curve shows the learning curve for the neural network with two hidden neurons.

### 4.9.3 Sequence Encoding

To feed the neural network with sequence data the amino acids must be transformed into numerical values in the input layer. A large set of different encoding schemes exists. The most conventionally used is the *sparse* or *orthogonal* encoding scheme, where each amino acid is represented as a 20- or 21-bit binary string. Alanine is represented as 10000000000000000000 and cysteine as 01000000000000000000, $\cdots$, where the last digit is used to represent blank, N- and C-terminal positions in a sequence window, i.e., when a window extends one of the ends of the sequence. Other encoding schemes take advantage of the physical and chemical similarities between the different amino acids. One such encoding scheme is the BLOSUM encoding, where each amino acid is encoded as the 20 BLOSUM matrix values for replacing the amino acid [Nielsen et al., 2003]. A summary of other sequence encoding schemes can be found in [Baldi and Brunak, 2001].

| | Predicted positive | Predicted negative | Total |
|---|---|---|---|
| Actual positive | TP | FN | AP |
| Actual negative | FP | TN | AN |
| Total | PP | PN | N |

Table 4.2: Classification of predictions. TP: true positives (predicted positive, actual positive); TN: true negatives (predicted negative, actual negative); FP: false positives (predicted positive, actual negative); FN: false negatives (predicted negative, actual positive).

## 4.10 Performance Measures for Prediction Methods

A number of different measures are commonly used to evaluate the performance of predictive algorithms. These measures differ according to whether the performance of a real-valued predictor (e.g., binding affinities) or a classification is to be evaluated.

In almost all cases percentages of correctly predicted examples are not the best indicators of the predictive performance in classification tasks, because the number of positives often is much smaller than the number of negatives in independent test sets. Algorithms that underpredict a lot will therefore appear to have a high success rate, but will not be very useful.

We define a set of performance measures from a set of data with $N$ predicted values $p_i$ and $N$ actual (or target) values $a_i$. The value $p_i$ is found using a prediction method of choice, and the $a_i$ is the known corresponding target value. By introducing a threshold $t_a$, the $N$ points can be divided into actual positives $A_P$ (points with actual values $a_i$ greater than $t_a$) and actual negatives $A_N$. Similarly, by introducing a threshold for the predicted values $t_p$, the points can be divided into predicted positives $P_P$ and predicted negatives $P_N$. These definitions are summarized in table 4.2 and will in the following be used to define a series of different performance measures.

### 4.10.1 Linear Correlation Coefficient

The linear correlation coefficient, which is also called Pearson's $r$, or just the correlation coefficient, is the most widely used measure of the association between pairs of values [Press et al., 1992]. It is calculated as

$$c = \frac{\sum_i (a_i - \overline{a})(p_i - \overline{p})}{\sqrt{\sum_i (a_i - \overline{a})^2}\sqrt{\sum_i (p_i - \overline{p})^2}} , \tag{4.48}$$

where the overlined letters denote average values. This is one of the best measures of association, but as the name indicates it works best if the actual

and predicted values when plotted against each other fall roughly on a line. A value of 1 corresponds to a perfect correlation and a value of $-1$ to a perfect anticorrelation (when the prediction is high, the actual value is low). A value of 0 corresponds to a random prediction.

### 4.10.2 Matthews Correlation Coefficient

I f all the predicted and actual values only take one of two values (normally 0 and 1) the linear correlation coefficient reduces to the Matthews correlation coefficient [Matthews, 1975]

$$c = \frac{T_P T_N - F_P F_N}{\sqrt{(T_P + F_N)(T_N + F_P)(T_P + F_P)(T_N + F_N)}} = \frac{T_P T_N - F_P F_N}{\sqrt{A_P A_N P_P P_N}} \,. \tag{4.49}$$

As for the Pearson correlation, a value of 1 corresponds to a perfect correlation.

### 4.10.3 Sensitivity, Specificity

Four commonly used measures are calculated by dividing the true positives and negatives by the actual and predicted positives and negatives [Guggenmoos-Holzmann and van Houwelingen, 2000],

**Sensitivity** Sensitivity measures the fraction of the actual positives which are correctly predicted: $sens = \frac{TP}{AP}$.

**Specificity** Specificity denotes the fraction of the actual negatives which are correctly predicted: $spec = \frac{TN}{AN}$

**PPV** The positive predictive value (PPV) is the fraction of the predicted positives which are correct: $PPV = \frac{TP}{PP}$.

**NPV** The negative predictive value (NPV) stands for the fraction of the negative predictions which are correct: $NPV = \frac{TN}{PN}$.

### 4.10.4 Receiver Operator Characteristics Curves

One problem with the above measures (except Pearson's $r$) is that a threshold $t_p$ must be chosen to distinguish between predicted positives and negatives. When comparing two different prediction methods, one may have a better Matthews correlation coefficient than the other. Alternatively, one may have a higher sensitivity or a higher specificity. Such differences may be due to the choice of thresholds and in that case the two prediction methods may

| Rank | Prediction | Actual | TPP | FPP | Area |
|---|---|---|---|---|---|
| 1 | 0.1 | 1 | 0.33 | 0 | 0 |
| 2 | 0.3 | 0 | 0.33 | 0.5 | 0.17 |
| 3 | 0.35 | 1 | 0.66 | 0.5 | 0.17 |
| 4 | 0.7 | 1 | 1.00 | 0.5 | 0.17 |
| 5 | 0.88 | 0 | 1.00 | 1 | **0.67** |

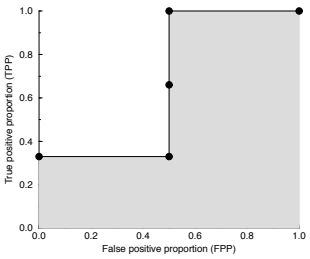



Figure 4.15: Calculation of a ROC curve. The table on the left side of the figure indicates the steps involved in constructing the ROC curve. The pairs of predicted and actual values must first be sorted according to the predicted value. The value in the lower right corner is the $A_{ROC}$ value. In the right panel of the figure is shown the corresponding ROC curve.

be rendered identical if the threshold for one of the methods is adjusted. To avoid such artifacts a nonparametric performance measure such as a receiver operator characteristics (ROC) curve is generally applied.

The ROC curve is constructed by using different values of the threshold $t_p$ to plot the false-positive proportion $FPP = F_P/A_N = F_P/(F_P + T_N)$ on the $x$-axis against the true positive proportion $TPP = T_P/A_P = T_P/(T_P + F_N)$ on the $y$-axis [Swets, 1988]. Figure 4.15 shows an example of how to calculate a ROC curve and the area under the curve, $A_{ROC}$, which is a measure of predictive performance. An $A_{ROC}$ value close to 1 indicates again a very good correlation; a value close to 0 indicates a negative correlation and a value of 0.5, no correlation. A general rule of thumb is that an $A_{ROC}$ value $> 0.7$ indicates a useful prediction performance, and a value $> 0.85$ a good prediction. $A_{ROC}$ is indeed a robust measure of predictive performance. Compared with the Matthews correlation coefficient, it has the advantage that it is independent of the choice of $t_p$. It is still, however, dependent on the choice of a threshold $t_a$ for the actual values. Compared with Pearson's correlation $r$ it has the advantage that it is nonparametric, i.e., that the actual value of the predictions is not used in the calculations, only their ranks. This is an advantage in situations where the predicted and actual values are related by a nonlinear function.

## 4.11 Clustering and Generation of Representative Sets

When training a bioinformatical prediction method, one very important initial step is to generate representative sets. If the data used to train, for instance, a neural network have many very similar data examples, the network will not be trained in an optimal manner. The reason for this is first of all that the network will focus on learning the data that are repeated and thereby get a lower ability to generalize. The other equally important point is that the performance of the prediction method will be overestimated, since the data in the training and test sets will be very alike.

Generating a representative set from a data set is therefore a very important part of the development of a prediction method. The general idea behind generation of representative sets is to exclude redundant data. In making a representative set one also implicitly makes a clustering since all data points which were removed because of similarity to another data point can be said to define a cluster.

In sequence analysis a number of algorithms exist for selecting a representative subset from a set of data points. This is generally done by keeping only one of two very similar data points. In order to do this a measure for similarity must be defined between two data points. For sequences this can, e.g., be percentage identity, alignment score, or significance of alignment score. Hobohm et al. [1992] have presented two algorithms for making a representative set from a list of data points D.

**Hobohm 1** Repeat for all data points on the list D:

- Add next data point in D to list of nonredundant data points N if it is not similar to any of the elements already on the list.

**Hobohm 2** Repeat until all sequences are removed from D:

- Add the data point S with the largest number of similarities to the non redundant set N.
- Remove data point S and all sequences similar to S from D.

Before applying the Hobohm 1 algorithm, the data points can be sorted according to some property. This will tend to maximize the average value of this property in the selected set because points higher on the list have less chance of being filtered out. The property can, e.g., be chosen to be the quality of the experimental determination of the data point. The Hobohm 2 algorithm aims at maximizing the size of the selected set by first removing the worst offenders, i.e., those with the largest number of neighbors. Hobohm 1 is faster than Hobohm 2 since it is in most cases not necessary to calculate the similarity between all pairs of data points.