Immunological Bioinformatics

Ole Lund
Morten Nielsen
Claus Lundegaard
Can Keşmir
Søren Brunak

The MIT Press
Cambridge, Massachusetts
London, England

This book was set in Lucida by the authors and was printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Immunological bioinformatics / Ole Lund .. [et al.].
p. cm. — (Computational molecular biology)
Includes bibliographical references and index.
ISBN 0-262-12280-4 (alk. paper)
1. Immunoinformatics. I. Lund, Ole. II. Series.
QR182.2.I46I465 2005
571.9'6'0285-dc22
2005042806

Contents

Preface **ix**

1 Immune Systems and Systems Biology **1**
- 1.1 Innate and Adaptive Immunity in Vertebrates 10
- 1.2 Antigen Processing and Presentation 11
- 1.3 Individualized Immune Reactivity 14

2 Contemporary Challenges to the Immune System **17**
- 2.1 Infectious Diseases in the New Millennium 17
- 2.2 Major Killers in the World 17
- 2.3 Childhood Diseases 20
- 2.4 Clustering of Infectious Disease Organisms 22
- 2.5 Biodefense Targets 28
- 2.6 Cancer 30
- 2.7 Allergy 30
- 2.8 Autoimmune Diseases 31

3 Sequence Analysis in Immunology **33**
- 3.1 Sequence Analysis 33
- 3.2 Alignments 34
- 3.3 Multiple Alignments 50
- 3.4 DNA Alignments 52
- 3.5 Molecular Evolution and Phylogeny 53
- 3.6 Viral Evolution and Escape: Sequence Variation 55
- 3.7 Prediction of Functional Features of Biological Sequences 59

4 Methods Applied in Immunological Bioinformatics **67**
- 4.1 Simple Motifs, Motifs and Matrices 67
- 4.2 Information Carried by Immunogenic Sequences 70
- 4.3 Sequence Weighting Methods 73
- 4.4 Pseudocount Correction Methods 75

4.5 Weight on Pseudocount Correction 77
4.6 Position Specific Weighting 77
4.7 Gibbs Sampling 78
4.8 Hidden Markov Models 82
4.9 Artificial Neural Networks 89
4.10 Performance Measures for Prediction Methods 97
4.11 Clustering and Generation of Representative Sets 100

5 DNA Microarrays in Immunology 101
5.1 DNA Microarray Analysis 101
5.2 Clustering 104
5.3 Immunological Applications 106

6 Prediction of Cytotoxic T Cell (MHC Class I) Epitopes 109
6.1 Background and Historical Overview of Methods for Peptide MHC Binding Prediction 110
6.2 MHC Class I Epitope Binding Prediction Trained on Small Data Sets 112
6.3 Prediction of CTL Epitopes by Neural Network Methods 118
6.4 Summary of the Prediction Approach 131

7 Antigen Processing in the MHC Class I Pathway 133
7.1 The Proteasome 133
7.2 Evolution of the Immunosubunits 135
7.3 Specificity of the (Immuno)Proteasome 137
7.4 Predicting Proteasome Specificity 141
7.5 Comparison of Proteasomal Prediction Performance 145
7.6 Escape from Proteasomal Cleavage 147
7.7 Post-Proteasomal Processing of Epitopes 148
7.8 Predicting the Specificity of TAP 151
7.9 Proteasome and TAP Evolution 152

8 Prediction of Helper T Cell (MHC Class II) Epitopes 155
8.1 Prediction Methods 156
8.2 The Gibbs Sampler Method 157
8.3 Further Improvements of the Approach 170

9 Processing of MHC Class II Epitopes 173
9.1 Enzymes Involved in Generating MHC Class II Ligands 174
9.2 Selective Loading of Peptides to MHC Class II Molecules 177
9.3 Phylogenetic Analysis of the Lysosomal Proteases 178
9.4 Signs of the Specificities of Lysosomal Proteases on MHC Class II Epitopes 180

9.5 Predicting the Specificity of Lysosomal Enzymes 180

10 B Cell Epitopes 185
10.1 Affinity Maturation 186
10.2 Recognition of Antigen by B cells 189
10.3 Neutralizing Antibodies 199

11 Vaccine Design 201
11.1 Categories of Vaccines 202
11.2 Polytope Vaccine: Optimizing Plasmid Design 205
11.3 Therapeutic Vaccines 207
11.4 Vaccine Market 211

12 Web-Based Tools for Vaccine Design 213
12.1 Databases of MHC Ligands 213
12.2 Prediction Servers 215

13 MHC Polymorphism 221
13.1 What Causes MHC Polymorphism? 221
13.2 MHC Supertypes 223

14 Predicting Immunogenicity: An Integrative Approach 241
14.1 Combination of MHC and Proteasome Predictions 242
14.2 Independent Contributions from TAP and Proteasome Predictions 243
14.3 Combinations of MHC, TAP, and Proteasome Predictions 245
14.4 Validation on HIV Data Set 249
14.5 Perspectives on Data Integration 250

References 252

Chapter 3

Sequence Analysis in Immunology

3.1 Sequence Analysis

The concept of protein families is based on the observation that, while there are a huge number of different proteins, most of them can be grouped, on the basis of similarities in their sequences, into a limited number of families. Proteins or protein domains belonging to a particular family generally share functional attributes and are derived from a common ancestor, and will most often be the result of gene duplication events.

It is apparent, when studying protein sequence families, that some regions have been more conserved than others during evolution. These regions are generally important for the function of a protein and/or the maintenance of its three-dimensional structure, or other features related to its localization or modification. By analyzing constant and variable properties of such groups of similar sequences, it is possible to derive a signature for a protein family or domain, which distinguishes its members from other unrelated proteins. Here we mention some examples of such domains that are essential to the immune response.

The immunoglobulin-like (Ig-like) protein domain is a domain of approximately 100 residues with a fold which consists of seven to nine antiparallel β strands. These β strands form a β-sandwich structure, consisting of three or four antiparallel β strands on each side of the barrel, connected by a sulfide bridge. The Ig-like domain is of special importance for the immune system. In addition to immunoglobulin, T cell receptor and MHC molecules carry Ig-like domains, i.e., the main players of the adaptive immune system have all Ig-like

domains. This is not a coincidence: the unique structure of this domain allows for maximum flexibility to interact with other molecules. This property makes the Ig-like domain one of the most widespread protein modules in the animal kingdom. This module has been observed in a large group of related proteins that function in cell-cell interactions or in the structural organization and regulation of muscles. The proteins in the Ig-like family consist of one or more of these domains.

Toll-like receptors (TLRs) are a family of pattern recognition receptors that are activated by specific components of microbes and certain host molecules. They constitute the first line of defense against many pathogens and play a crucial role in the function of the innate immune system.

That the field of immunology is almost as big, dispersed, and complicated as all the rest of the biology put together is exemplified by the fact that all the different fields of bioinformatics and sequence analysis are applied to immunological problems. Sequence alignment, structural biology, machine learning and predictive systems, pattern recognition, DNA microarray analysis, and integrative systems biology are all important tools in the research of the different aspects of the immune system and its interaction with pathogens.

3.2 Alignments

Sequence alignment is the oldest but probably the single most important tool in bioinformatics. Being one of the basic techniques within sequence analysis, alignment is, though, far from simple, and the analytic tools (i.e., the computer programs) are still not perfect. Furthermore, the question of which method is optimal in a given situation strongly depends on which question we want the answer to. The most common questions are: How similar (different) are this group of sequences, and which sequences in a database are similar to a specific query sequence. The reasoning behind the questions might, however, be important for the choice of algorithmic solution. Why do we want to know this? Are we searching for the function of a protein/gene, or do we want to obtain an estimate of the evolutionary history of the protein family? Issues like the size of database to search, and available computational resources might also influence our selection of a tool.

3.2.1 Ungapped Pairwise Alignments

From the early days of protein and DNA sequencing it was clear that sequences from highly related species were highly similar, but not necessarily identical. Aligning very closely related sequences is a trivial task and can be done manually (figure 3.1 A). In cases where genes are of different sizes and the similarity

A

```
               10        20        30        40        50        60        70
humanD MSEKKQPVDLGLLEEDDEFEEFPAEDWAGLDEDEDAHVWEDNWDDDNVEDDFSNQLRAELEKHGYKMETS
       :::::::::::::::::::::.:::::::::::::::::::::::::::::::::::::::::::::::
gi|457 MSEKKQTVDLGLLEEDDEFEEFPAEDWTGLDEDEDAHVWEDNWDDDNVEDDFSNQLRAELEKHGYKMETS
               10        20        30        40        50        60        70
```

B

```
                   10        20        30        40        50        60        70
humanD ----MSEKKQPVDLGLLEEDDEFEEFPAEDWAGLDEDEDAHVWEDNWDDDNVEDDFSNQLRAELEKHGYKMETS
           ....:...:::::::::::::::::::..:::.........::....:..::.........
Anophe MSDKENKDKPKLDLGLLEEDDEFEEFPAEDWAGNKEDEEELSVWEDNWDDDNVEDDFNQQLRAQLEKHK-----
               10        20        30        40        50        60        70
```

Figure 3.1: A) The human proteasomal DSS1 subunit aligned against the zebra fish homolog using the identity matrix. B) The human proteasomal DSS1 subunit aligned to the mosquito homolog.

is less, alignments become more difficult to construct. In such cases it is also of great value to have a graduation of how related sequences are, i.e., a scoring scheme. The simplest scoring is the relative amount of identical entities, also called % identity, or %ID. This simple approach is actually too simple as ,e.g., amino acids share many physical-chemical properties, which means that they can more easily be exchanged than very unrelated amino acids. This means that a scoring system that scores different substitutions differently, a substitution matrix, is a much better approach. The most useful concept has been to estimate how often a given amino acid is exchanged for another in already aligned similar sequences. The most used are the *percentage accepted mutations* (PAM) matrix [Dayhoff et al., 1978] and the *blocks substitution matrix* (BLOSUM) [Henikoff and Henikoff, 1992].Mutations between different types of nucleotides or amino acids is not the only changes that appears in sequences during the evolution. The sequences can also loose or gain sequence entities (deletions or insertions, respectively). This also must affect a similarity score, but for simplicity these complications are left to later sections. The simplest way to calculate an alignment score is to make all the possible overlaps between two sequences, and sum the number of identical amino acids in the two sequences (ungapped alignment, figure 3.1 B).

Sequence alignment is essential to the *comparative immunology* field. The main research line in this field (so far) is to discover origins of the adaptive immune system. Thanks to the homology assessments using sequence alignments with mammalian equivalents of T cell receptors, MHC genes, cytokines, and antibodies, we now know that the adaptive immune system is well developed in the oldest jawed vertebrates, the sharks [Pasquier and Flajnik, 1999]. However, whether or not jawless invertebrates were in possession of such

adaptive immunity remains unresolved. The lamprey, which along with its cousin, the hagfish, is the only surviving jawless vertebrate, give immunologists a chance to pinpoint crucial aspects of the origin of the adaptive immune system. So far the search for antibodies, T cell receptors, and genes coding for MHC molecules has failed in these organisms. Recently, however, Pancer et al. [2004] have identified a set of uniquely diverse proteins that are only expressed by lamprey lymphocytes and named them variable lymphocyte receptors (VLRs). The sequence analysis of these proteins has revealed that the VLRs consist of multiple leucine-rich repeat (LRR) modules and an invariant stalk region that is attached to the lymphocyte plasma membrane. The remarkable VLR diversity derives from the variation in sequence and number of the LRR modules. The mature VLRs are thus generated through a process of somatic DNA rearrangement in lymphocytes. These results suggest a novel mechanism that does not involve recombinant-activating genes to generate the large diversity that an adaptive immune system is based upon.

3.2.2 Scoring Matrices

Dayhoff et al. [1978] calculated the original PAM matrices using a database of changes in groups of closely related proteins. From these changes they derived the accepted types of mutations. Each change was entered into a matrix listing all the possible amino acid changes. The relative mutability of different amino acids was also calculated, i.e., how often a given amino acid is changed to any other. The information about the individual kinds of mutations, and about the relative mutability of the amino acids were then combined into one "mutation probability matrix."

The rows and columns of this matrix represent amino acid substitution pairs, i.e., the probability that the amino acid of the column will be replaced by the amino acid of the row after a given evolutionary interval. A matrix with an evolutionary distance of 0 PAMs would have only 1s on the main diagonal and 0s elsewhere. A matrix with an evolutionary distance of 1 PAM would have numbers very close to 1 in the main diagonal and small numbers off the main diagonal. One PAM would correspond to roughly a 1% divergence in a protein (one amino acid replacement per hundred). Assuming that proteins diverge as a result of accumulated, uncorrelated, mutations a mutational probability matrix for a protein sequence that has undergone N percent accepted mutations, a PAM-N matrix, can be derived by multiplying the PAM-1 matrix by itself N times. The result is a whole family of scoring matrices. Dayhoff et al. [1978], imperically, found that for weighting purposes a 250 PAM matrix works well. This evolutionary distance corresponds to 250 substitutions per hundred residues (each residue can change more than once). At this distance

A

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z	X
A	2	-2	0	0	-2	0	0	1	-1	-1	-2	-1	-1	-3	1	1	1	-6	-3	0	0	0	0
R	-2	6	0	-1	-4	1	-1	-3	2	-2	-3	3	0	-4	0	0	-1	2	-4	-2	-1	0	-1
N	0	0	2	2	-4	1	1	0	2	-2	-3	1	-2	-3	0	1	0	-4	-2	-2	2	1	0
D	0	-1	2	4	-5	2	3	1	1	-2	-4	0	-3	-6	-1	0	0	-7	-4	-2	3	3	-1
C	-2	-4	-4	-5	12	-5	-5	-3	-3	-2	-6	-5	-5	-4	-3	0	-2	-8	0	-2	-4	-5	-3
Q	0	1	1	2	-5	4	2	-1	3	-2	-2	1	-1	-5	0	-1	-1	-5	-4	-2	1	3	-1
E	0	-1	1	3	-5	2	4	0	1	-2	-3	0	-2	-5	-1	0	0	-7	-4	-2	3	3	-1
G	1	-3	0	1	-3	-1	0	5	-2	-3	-4	-2	-3	-5	0	1	0	-7	-5	-1	0	0	-1
H	-1	2	2	1	-3	3	1	-2	6	-2	-2	0	-2	-2	0	-1	-1	-3	0	-2	1	2	-1
I	-1	-2	-2	-2	-2	-2	-2	-3	-2	5	2	-2	2	1	-2	-1	0	-5	-1	4	-2	-2	-1
L	-2	-3	-3	-4	-6	-2	-3	-4	-2	2	6	-3	4	2	-3	-3	-2	-2	-1	2	-3	-3	-1
K	-1	3	1	0	-5	1	0	-2	0	-2	-3	5	0	-5	-1	0	0	-3	-4	-2	1	0	-1
M	-1	0	-2	-3	-5	-1	-2	-3	-2	2	4	0	6	0	-2	-2	-1	-4	-2	2	-2	-2	-1
F	-3	-4	-3	-6	-4	-5	-5	-5	-2	1	2	-5	0	9	-5	-3	-3	0	7	-1	-4	-5	-2
P	1	0	0	-1	-3	0	-1	0	0	-2	-3	-1	-2	-5	6	1	0	-6	-5	-1	-1	0	-1
S	1	0	1	0	0	-1	0	1	-1	-1	-3	0	-2	-3	1	2	1	-2	-3	-1	0	0	0
T	1	-1	0	0	-2	-1	0	0	-1	0	-2	0	-1	-3	0	1	3	-5	-3	0	0	-1	0
W	-6	2	-4	-7	-8	-5	-7	-7	-3	-5	-2	-3	-4	0	-6	-2	-5	17	0	-6	-5	-6	-4
Y	-3	-4	-2	-4	0	-4	-4	-5	0	-1	-1	-4	-2	7	-5	-3	-3	0	10	-2	-3	-4	-2
V	0	-2	-2	-2	-2	-2	-2	-1	-2	4	2	-2	2	-1	-1	-1	0	-6	-2	4	-2	-2	-1
B	0	-1	2	3	-4	1	3	0	1	-2	-3	1	-2	-4	-1	0	0	-5	-3	-2	3	2	-1
Z	0	0	1	3	-5	3	3	0	2	-2	-3	0	-2	-5	0	0	-1	-6	-4	-2	2	3	-1
X	0	-1	0	-1	-3	-1	-1	-1	-1	-1	-1	-1	-1	-2	-1	0	0	-4	-2	-1	-1	-1	-1

B

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z	X
A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0	-2	-1	0
R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3	-1	0	-1
N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3	3	0	-1
D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3	4	1	-1
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1	-3	-3	-2
Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2	0	3	-1
E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2	1	4	-1
G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3	-1	-2	-1
H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3	0	0	-1
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3	-3	-3	-1
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1	-4	-3	-1
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	0	-1	-3	-2	-2	0	1	-1
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1	-3	-1	-1
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4	-2	-2	1	3	-1	-3	-3	-1
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7	-1	-1	-4	-3	-2	-2	-1	-2
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2	0	0	0
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0	-1	-1	0
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3	-4	-3	-2
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1	-3	-2	-1
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4	-3	-2	-1
B	-2	-1	3	4	-3	0	1	-1	0	-3	-4	0	-3	-3	-2	0	-1	-4	-3	-3	4	1	-1
Z	-1	0	0	1	-3	3	4	-2	0	-3	-3	1	-1	-3	-1	0	-1	-3	-2	-2	1	4	-1
X	0	-1	-1	-1	-2	-1	-1	-1	-1	-1	-1	-1	-1	-2	0	0	-2	-1	-1	-1	-1	-1	-1

Figure 3.2: Substitution matrices. A) PAM250. B) BLOSUM62.

only one amino acid in five remains unchanged so the percent divergence has increased to roughly 80%. To avoid working with very small numbers the matrices actually used in sequence comparisons is logodds matrices. The odds matrix is constructed by taking the elements of the previous matrix and divide each component by the frequency of the replacement residue. In this way each component now gives the odds of replacing a given amino acid with another specified amino acid. Finally the log of this matrix is used as the weights in the matrix. In this it is now possible to sum up the scores for all positions to obtain the final alignment score. The PAM250 matrix is shown in Figure 3.2.

A

```
                    10         20         30         40         50         60         70
humanD -----MSEKKQPVDLGLLEEDDEFEEFPAEDWAGLDEDEDAHVWEDNWDDDNVEDDFSNQLRAELEKHGYKMETS
             ..: .     : :.:.  :    .  .. ...:.  ::::::::::::::..::::.::::
Anophe MSDKENKDKPKLDLGLLEEDDEFEEFPAEDWAGNKEDEEELSVWEDNWDDDNVEDDFNQQLRAQLEKHK------
               10         20         30         40         50         60
```

B

```
                   10         20         30         40         50         60         70
humanD ----MSEKKQPVDLGLLEEDDEFEEFPAEDWAGLDEDEDAHVWEDNWDDDNVEDDFSNQLRAELEKHGYKMETS
           ....:...:::::::::::::::::::::..:::.........::....:..::.........
Anophe MSDKENKDKPKLDLGLLEEDDEFEEFPAEDWAGNKEDEEELSVWEDNWDDDNVEDDFNQQLRAQLEKHK-----
               10         20         30         40         50         60
```

Figure 3.3: (A) The human proteasomal subunit aligned to the mosquito homolog using the BLOSUM50 matrix. (B) The human proteasomal subunit aligned to the mosquito homolog using identity scores.

The BLOSUM matrix, described by Henikoff and Henikoff [1992], is another widely used amino acid substitution matrix. To calculate this, only very related blocks of amino acid sequences (conserved blocks) are considered. Originally these were taken from the BLOCKS database of prealigned sequence families [Henikoff and Henikoff, 1991]. Now the blocks are split up further in clusters, each containing the parts of the alignments that are more than X% conserved. The use of these clusters leads to a BLOSUMX matrix. That is, using clusters of down to 50% identities gives a BLOSUM50 matrix, and so forth. For every sequence in each cluster each position is compared to the corresponding position in each sequence in every other cluster. Since it is the *pairwise* number of frequencies that is calculated, the sum of all the substitutions is divided by the number of comparisons. In this way the result is the weighted probability that a given amino acid is exchanged for every other amino acid. In the final matrix, actually, the log ratio of the probability is further scaled so that the BLOSUM50 matrix is in thirds of bits, and the BLOSUM62 matrix is given in half-bits. The BLOSUM62 matrix is shown in figure 3.2.

Since the initial PAM1 matrix is made by very similar sequences, the evolutionary distances between those are very short, and most changes captured will be single base mutations leading to particular types of amino acid substitutions, while substitutions requiring more than one base mutation will be very rare. Even the calculations made to expand this matrix to longer evolution time cannot compensate for this [Gonnet et al., 1992] and therefore the BLOSUM matrices perform better when used for further distance alignment. The matrices are in a format where you can sum up the scores for each match to obtain a total alignment score, and the alignment resulting in the highest score is then the optimal one.

3.2.3 Gap Penalties

Using the BLOSUM50 matrix to align mosquito and human proteasomeal subunits (figure 3.3A) gives a slightly different alignment than just using amino acid identities (figure 3.3B). These two different alignments also reveal that there are two parts of the proteins with a high number of identical amino acids, but without inserting or deleting letters in one of the sequences they cannot be aligned simultaneously. This leads obviously to the necessity of inserting gaps in the alignments.

A gap in one sequence represents an insertion in the other sequence. First, to avoid having gaps all over the alignment these have to be penalized just like unmatching amino acids. This penalty (i.e., the probability that a given amino acid will be deleted in another related sequence) cannot be derived from the database alignments used to create the PAM and BLOSUM matrices, since these are ungapped alignments. Instead, a general gap insertion penalty is determined, usually empirically, and is often lower than the lowest match score. Having only one score for any gap inserted is called a linear gap cost, and will lead to the same total penalty for three single gaps at three different positions in the alignment as having a single stretch of three gaps. This does not make sense biologically, however, since insertions and deletions often involve a longer stretch of DNA in a single event. For this reason two different gap penalties are usually included in the alignment algorithms: one penalty for having a gap at all (gap opening penalty), and another, smaller penalty, for extending already opened gaps. This is called an affine gap penalty and is actually a compromise between the assumption that the insertion, or deletion, is created by one or more events. Furthermore, it is possible to let gaps appended at the ends of the sequences not to have a penalty, since insertions at the ends will have a much greater chance of not disrupting the function of a protein. For a more careful discussion of how to set gap penalties, see Vingron and Waterman [1994].

3.2.4 Alignment by Dynamic Programming

Introducing gaps greatly increases the number of different comparisons between two sequences and in the general case it is impossible to do them all. To compensate for that, several shortcut optimization schemes have been invented. One of the earliest schemes was developed by Needleman and Wunsch [1970] and works for global alignments, i.e., alignments covering all residues in both sequences. As an example, it is here described how to align two very short sequence stretches taken from our previous proteasome alignment. For simplicity, we will use the identity matrix (match=1, mismatch=-1) and a linear gap penalty of -2. Using the Needleman-Wunsch approach

Score matrix

		D		E		D		E		D		A		H		V		W	
	0																		
K																			
E																			
D																			
E																			
E																			
E																			
L																			
S																			
V																			
W																			

Trace Matrix

		D		E		D		E		D		A		H		V		W	
	END																		
K																			
E																			
D																			
E																			
E																			
E																			
L																			
S																			
V																			
W																			

Figure 3.4: Dynamic programming, global alignment. Step 1.

Score matrix

		D		E		D		E		D		A		H		V		W	
	0		-2																
K																			
	-2																		
E																			
D																			
E																			
E																			
E																			
L																			
S																			
V																			
W																			

Trace Matrix

		D		**E**		**D**		**E**		**D**		**A**		**H**		**V**		**W**	
	END	-	right																
K	|																	
	up																		
E																			
D																			
E																			
E																			
E																			
L																			
S																			
V																			
W																			

Figure 3.5: Dynamic programming, global alignment. Step 2.

Score matrix

		D		E		D		E		D		A		H		V		W	
	0		-2																
K																			
	-2		-1																
E																			
D																			
E																			
E																			
E																			
L																			
S																			
V																			
W																			

Trace Matrix

		D		E		D		E		D		A		H		V		W	
	END	-	right																
K	|	\																	
	up		diagonal																
E																			
D																			
E																			
E																			
E																			
L																			
S																			
V																			
W																			

Figure 3.6: Dynamic programming, global alignment. Step 3.

Score matrix

		D		E		D		E		D		A		H		V		W	
	0		-2		-4		-6		-8		-10		-12		-14		-16		-18
K																			
	-2		-1		-3		-5		-7		-9		-11		-13		-15		-17
E																			
	-4		-3		0		-2		-4		-6		-8		-10		-12		-14
D																			
	-6		-3		-2		1		-1		-3		-5		-7		-9		-11
E																			
	-8		-5		-2		-1		2		0		-2		-4		-6		-8
E																			
	-10		-7		-4		-3		0		1		-1		-3		-5		-7
E																			
	-12		-9		-6		-5		-2		-1		0		-2		-4		-6
L																			
	-14		-11		-8		-7		-4		-3		-2		-1		-3		-5
S																			
	-16		-13		-10		-9		-6		-5		-4		-3		-2		-4
V																			
	-18		-15		-12		-11		-8		-7		-6		-5		-2		-3
W																			
	-20		-17		-14		-13		-10		-9		-8		-7		-4		-1

Trace Matrix

		D		**E**		**D**		**E**		**D**		**A**		**H**		**V**		**W**	
	END	-	left	-	left	-	left	-	left	-	left	-	left	-	left	-	left	-	left
K	|	\																	
	up		diagonal	-	left	-	left	-	left	-	left	-	left	-	left	-	left	-	left
E	|		|	\															
	up		up		diagonal		left	-	left	-	left		left	-	left	-	left	-	left
D	|	\	|		|	\													
	up		diagonal		up		diagonal	-	left	-	left		left	-	left	-	left	-	left
E	|		|	\			|	\											
	up		up		diagonal		up		diagonal	-	left		left	-	left	-	left	-	left
E	|		|		|		|		|	\								
	up		up		up		up		up		diagonal		left	-	left	-	left	-	left
E	|		|		|		|		|		\							
	up		up		up		up		up		up		diagonal		left	-	left	-	left
L	|		|		|		|		|		|	\					
	up		up		up		up		up		up		up		diagonal	-	left	-	left
S	|		|		|		|		|		|		|	\			
	up		up		up		up		up		up		up		up		diagonal	-	left
V	|		|		|		|		|		|		|	\		\	
	up		up		up		up		up		up		up		up		diagonal		diagonal
W	|		|		|		|		|		|		|		|	\	
	up		up		up		up		up		up		up		up		up		diagonal

Figure 3.7: Dynamic programming, global alignment, final matrices (Needleman-Wunsch).

[Needleman and Wunsch, 1970], we first define two identical matrices with the same number of columns as residues in sequence 1 and as many rows as residues in sequence 2 One matrix is used to keep track of the scores and another to keep track of our route (see figures 3.4-3.7).

- **Step 1 (figure 3.4):** In the upper left field of the score matrix is written the score 0. This is the score before having aligned anything. From this field we can move in three directions: Down corresponds to inserting a gap in sequence 1, left to inserting a gap in sequence 2 and diagonal to making a match. Accordingly, a step to the right is -2, a step down is -2, and a diagonal step is $+1$ if the residues are identical, otherwise -1.
- **Step 2 (figure 3.5):** With the limits of the steps, we can easily fill in the first row and the first column of the matrix, since these fields can only be reached from one direction. So in the score matrix we write -2 in field 0,1, since this step corresponds to inserting a gap. In the trace matrix we then write *up* in field 0,1 since this was the direction we were coming from. In field 1,0 we write -2 in the score matrix and *left* in the trace matrix.
- **Step 3 (figure 3.6):** Now we would like to calculate the score of field 1,1. Coming from the left we had -2 in the previous field $(0,1)$ and will have to add -2 for making a move to the right, inserting a gap in the *other* sequence, resulting in a score of -4. We do likewise if we would come down from field 1,0. We can now also make a diagonal move which means a match between the two first residues. In this example they are not identical and the match will have the score -1. Since we came from 0,0 with the score 0 the match case will result in -1. So we have the possibility to make three different moves resulting in a score of -4, -4, or -1, respectively. We now select the move resulting in the highest score (i.e., -1), and we write this score in field 1,1 in the score matrix. In the trace matrix we write *diagonal* in field 1,1 since this was the type of move made to reach this score.
- **Final steps:** Steps 2 and 3 are repeated until both matrices are filled out (figure 3.7). In the case that two different moves to a field result in the same score, we select the move *coming* from the highest previous score to write in the trace matrix. At any field, we will finally have a score. This score is then the maximal alignment score you can get coming from the upper left diagonal and to the position in the sequences matching that field.

When the matrices are all filled out, the final alignment score is in the lower right corner of the score matrix. In the above example the final alignment score

is then −1. The score matrix has now served its purpose and is discarded, and the alignment is reconstructed using the trace matrix. To reconstruct the alignment start in the lower right corner of the final trace matrix (figure 3.7). Following the directions written in the fields, the alignment is now reconstructed backward. Here *diagonal* means a match between the two last residues in each sequence (W match W), and a move diagonal up-left. Next field: *diagonal*, i.e., V match V and a move diagonal up-left. The present field value is now *up*: This means that we introduce a gap in the first sequence to match S in the second sequence and then move one field up in the trace matrix. The rest of the trace is all diagonal, which means no gaps, and the resulting alignment will be

```
DEDEDAH-VW
KEDEEELSVW
```

This way to produce an alignment is called dynamic programming, and is still used in major alignment software packages (e.g., the ALIGN tool in the FASTA package uses the Needleman-Wunsch algorithm for global alignments). To illustrate that there *are* differences in the resulting alignments according to which scoring scheme is used, the above alignment using the BLOSUM62 matrix in figure 3.2 and a linear gap penalty of −9 results in the following alignment

```
DEDEDA-HVW
KEDEEELSVW
```

So the optimal alignment is only optimal using the chosen substitution scores and gap penalties, and there is no exact way to tell in a particular example if one set of scores gives a more “correct” alignment than another set of scores.

3.2.5 Local Alignments and Database Searches

The global alignment scheme described above is very good for comparing and analyzing the relationship between two selected proteins. Proteins, however, are often comprised of different domains, where each domain may be evolutionarily related to a different set of sequences. Thus when it comes to *searching* for sequences it is more beneficial to only look at the parts of the sequences that actually are related. A search is actually to make pairwise alignment of your query sequence to all the sequences in the database, and order the resulting alignments by the alignment score. For this purpose Smith and Waterman [1981] further developed the dynamic programming approach. The Smith-Waterman algorithm is like Needleman-Wunsch, except that the traces only continue as long as the scores are positive, Whenever a score becomes negative it is set to 0 and the corresponding trace is empty. Using the BLOSUM62 substitution matrix and a linear gap penalty of −9, the score and trace

Score matrix

		D		E		D		E		D		A		H		V		W	
	0		0		0		0		0		0		0		0		0		0
K																			
	0		0		1		0		1		0		0		0		0		0
E																			
	0		2		5		3		5		3		0		0		0		0
D																			
	0		6		4		11		5		11		2		0		0		0
E																			
	0		2		11		6		16		7		10		2		0		0
E																			
	0		2		7		13		11		18		9		10		1		0
E																			
	0		2		7		9		18		13		17		9		8		0
L																			
	0		0		0		3		9		14		12		14		10		6
S																			
	0		0		0		0		3		9		15		11		12		7
V																			
	0		0		0		0		0		0		9		12		15		9
W																			
	0		0		0		0		0		0		0		7		9		26

Trace Matrix

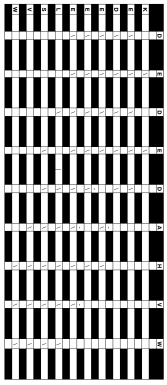


Figure 3.8: Dynamic programming, local alignment, final matrices (Smith-Waterman).

matrices will appear as in Figure 3.8. Now the backtrace of the optimal local alignment starts in the field with the highest score. There might be several equally good alignments, and there are several ways to deal with that, depending on what the goal is. If the two equally good alignments differ in length, one might, e.g., chose the longer. In this example the highest score is 26. This is accidentally again in the lower right corner so the backtrace will begin here. The backtrace will reveal that the local alignment look like this:

```
DEDEDAHVW
EDEEELSVW
```

BLAST The dynamic programming algorithm has the strength that it ensures that the optimal alignment, will always be found, given specific gap penalties and substitution scores. However, even with present-day computerpower this algorithm is far too slow to search the ever-increasing sequence databases of today. For this reason several shortcuts have been made, and one of the most successful is implemented in the widely-used alignment package, BLAST [Altschul et al., 1990, 1997, Altschul and Gish, 1996].

The basic BLAST algorithm consists of 3 steps:

1. **Make a list of words:** A list of neighbor words that have a score of at least T (default 11 for proteins) is made for each n-mer in the query sequence. Per default n=3 for proteins and n=11 for DNA. Any word in the query sequence that scores positive with itself may also be included.

2. **Search the database for the words on the list:** The database is scanned for hits to any of the N words on the list.

3. **Extend hits:** The first version of BLAST extended every hit it found. The newer version requires two nonoverlapping hits within a distance A (default 40) of each other before it extends a hit. The extension is only made until the score has dropped X (default 7) below the best score seen so far. This corresponds to saying this route looks so bad that there is no point in continuing in this direction. The locally optimal alignments are called high-scoring segment pairs (HSPs). If the score of an HSP is above a threshold S_g (default 22 bits) a gapped extension is attempted using dynamic programming. To speed the calculations this phase is only continued until the score falls X_g below the best score seen so far.

3.2.6 Expectation Values

When aligning two sequences it is not clear if a given score is really significant (i.e., might occur by chance by a certain probability). Such a measure can be

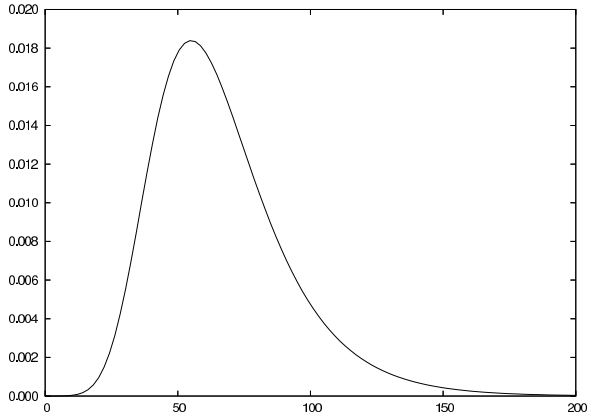


Figure 3.9: Distributions of scores, when aligning a sequence to a database of unrelated sequences.

obtained by aligning a great number of random sequences to the original sequence and from the resulting score distribution calculate the probability that a random sequence would result in a given score. This number is called the expectation-value, or E-value. The random sequences is obtained by shuffling the elements (nucleotides or amino acids) of the original sequence. In this way the score distribution will not be biased by a skewed amino acid distribution of the original sequence.

When searching through databases the question also arises whether a given alignment score confers a relationship between the two aligned regions or not. If we align a sequence to a database of all unrelated sequences and plot the alignment score against how many alignments will have that score we will get a curve like that in figure 3.9. This is called an extreme value distribution. We can from this distribution find out how often a given alignment-score will arise by chance. Thus the E-value is the theoretically expected number of false hits per sequence query, and a lower E-value means a more significant hit. Importantly, the E-value is dependent on the size of the database searched as the chance of getting a false hit rises as the database grows.

Different alignment programs use different approaches to calculate the E-value of a given database hit. FASTA actually makes all possible alignments, and returns a real distribution curve (figure 3.10) and calculates the E-value

```
       opt      E()
< 20     0     0:
  22     0     0:            one = represents 23 library sequences
  24     0     0:
  26     0     0:
  28     0     3:*
  30     0    16:*
  32     7    64:= *
  34    75   173:====   *
  36   240   354:===========    *
  38   569   586:=========================*
  40  1127   817:==================================*=============
  42  1379   999:============================================*================
  44  1277  1102:=============================================*========
  46  1183  1122:================================================*===
  48   914  1074:=======================================      *
  50   733   980:==============================          *
  52   753   862:================================    *
  54   661   736:============================  *
  56   516   615:====================   *
  58   536   505:====================*==
  60   365   409:================ *
  62   335   328:==============*
  64   273   261:===========*
  66   188   206:========*
  68   168   162:========*
  70   126   127:======*
  72   133    99:====*=
  74    88    77:===*
  76    68    60:==*
  78    56    47:==*
  80    41    36:=*
  82    41    28:=*
  84    34    22:*=
  86    16    17:*
  88    13    13:*           inset = represents 1 library sequences
  90    12    10:*
  92     6     8:*          :====== *
  94     4     6:*          :==== *
  96     3     5:*          :=== *
  98     4     4:*          :===*
 100     2     3:*          :==*
 102     0     2:*          : *
 104     0     2:*          : *
 106     1     1:*          :*
 108     2     1:*          :*=
 110     0     1:*          :*
 112     2     1:*          :*=
 114     0     0:           *
 116     0     0:           *
 118     0     0:           *
>120     0     0:           *
4113207 residues in 11951 sequences
 Expectation_n fit: rho(ln(x))= 5.3517+/-0.00135; mu= -2.1992+/- 0.077;
 mean_var=60.8388+/-13.111, Z-trim: 5  B-trim: 3 in 1/55
 Kolmogorov-Smirnov  statistic: 0.0520 (N=29) at  46
```

Figure 3.10: Distributions of scores, from FASTA alignments of a given sequence to all sequences in a specific database.

making a fit to this curve. BLAST, however, uses a premade empirical curve to assign E-values to each alignment returned from a database search.

PSI-BLAST As described earlier, the scoring matrices used somehow represent the general evolutionary trends for mutations. However, in reality, allowed mutations are very much dependent on, and constrained by their physical context. As an example, it could be possible to insert, delete, or exchange a number of different amino acids in a flexible loop on the surface of a protein and still preserve the overall structure and function of the protein.

		A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
1	I	-2	-4	-5	-5	-2	-4	-4	-5	-5	6	0	-4	0	-2	-4	-4	-2	-4	-3	4
2	K	-1	-1	-2	-2	-3	-1	3	-3	-2	-2	-3	4	-2	-4	-3	1	1	-4	-3	2
3	E	5	-3	-3	-3	-3	3	1	-2	-3	-3	-3	-2	-2	-4	-3	-1	-2	-4	-3	1
4	E	-4	-3	2	5	-6	1	5	-4	-3	-6	-6	-2	-5	-6	-4	-2	-3	-6	-5	-5
5	H	-4	2	1	1	-5	1	-2	-4	9	-5	-2	-3	-4	-4	-5	-3	-4	-5	1	-5
6	V	-3	0	-4	-5	-4	-4	-2	-3	-5	1	-2	1	0	1	-4	-3	3	-5	-3	5
7	I	0	-2	-4	1	-4	-2	-4	-4	-5	1	0	-2	0	2	-5	1	-1	-5	-3	4
8	I	-3	0	-5	-5	-4	-2	-5	-6	1	2	4	-4	-1	0	-5	-2	0	-3	5	-1
9	Q	-2	-3	-2	-3	-5	4	-1	3	5	-5	-3	-3	-4	-2	-4	2	-1	-4	2	-2
10	A	2	-4	-4	-3	2	-3	-1	-4	-2	1	-1	-4	-3	-4	1	2	3	-5	-1	1
11	E	-1	3	1	1	-1	0	1	-4	-3	-1	-3	0	3	-5	4	-1	-3	-6	-3	-1
12	F	-3	-5	-5	-5	-4	-4	-4	-1	-1	1	1	-5	2	5	-1	-4	-4	-3	5	2
13	Y	3	-5	-5	-6	3	-4	-5	-2	-1	0	-4	-5	-3	3	-5	-2	-2	-2	7	1
14	L	-1	-3	-4	-2	1	5	1	-1	-1	-1	1	-3	-3	1	-5	-1	-1	-2	3	-2
15	N	-1	-4	4	1	5	-3	-4	2	-4	-4	-4	-3	-2	-4	-5	2	0	-5	0	0
16	P	-2	4	-4	-4	-5	0	-3	3	2	-5	-4	0	-4	-3	0	1	-2	-1	5	-3
17	D	-3	-2	1	5	-6	-2	2	2	-1	-2	-2	-3	-5	-4	-5	-1	2	-6	-3	-4

Figure 3.11: Example of a PSSM.

The corresponding number of allowed substitutions would very probably be much more limited in the core — or in a secondary structure, rich — region of the protein. So if a *general* substitution matrix works well, a matrix representing the *specific* evolutionary trend for a given position in a given protein should work even better. As described by Altschul et al. [1997], this is actually the case.

In the PSI-BLAST approach, first an ordinary BLAST search on the basis of the BLOSUM62 matrix is performed against the database. Second, a position-specific scoring matrix (PSSM) is calculated as described in chapter 4. The matrix is calculated by considering the substitutions observed in pairwise alignments made between the query sequence and the hits that have an expectation value below a selected threshold. Now the calculated matrix (figure 3.11), as a representation of the query sequence, is used to search the database again. So when the alignment score matrix is filled out, we now look in the PSSM for a given position to find the match score between the PSSM and that particular amino acid in the database sequence. For example, if we want to match position 3 in the search sequence, a glutamic acid, to an alanine, the match score is 5. However, if we want to match position 4, also a glutamic acid, to an alanine, the match score is -4. This should illustrate the higher specificity of a PSSM as compared to ordinary substitution matrices.

3.3 Multiple Alignments

When looking at several related sequences, it is often useful and informative to look at all the sequences in one alignment (multiple alignment). The simplest approach is to align all the sequences, one by one, with a single selected "master sequence," and this is what can be obtained by programs like BLAST. However, these programs make only local alignments, and often gaps and in-

A

```
Drosophila_melanogaster       MSAPDKEKEKEKEETNNKSEDLGLLEEDDEFEEFPAEDFRVGDDEEELNVWEDNWDDDNVEDDFSQQLKAHLESKKMET
Anopheles_gambiae             ----------DKENKDKPKLDLGLLEEDDEFEEFPAEDWAGneDEEELSVWEDNWDDDNVEDDFNQQLRAQLEKHK---
Zebrafish                     ------------------QTVDLGLLEEDDEFEEFPAEDWTGLDEDEDAHVWEDNWDDDNVEDDFSNQLRAELE------
HUMAN                         --------------------DLGLLEEDDEFEEFPAEDWAGLDEDEDAHVWEDNWDDDNVEDDFSNQLRAELE------
MOUSE                         --------------------DLGLLEEDDEFEEFPAEDWAGLDEDEDAHVWEDNWDDDNVEDDFSNQLRAELE------
Xenopus_laevis                --------------------DLGLLEEDDEFEEFPTEDWTGFDEDEDTHVWEDNWDDDNVEDDFSNQLRAELE------
Saccharomyces_cerevisiae      ------------------------LEEDDEFEDFPIDTWANGETIkqTNIWEENWDDVEVDDDFTNELKAELDRYKRE-
Neurospora_crassa.            ----DAKSTEPKPEQPVTEKKTAVLEEDDEFEDFPVDDWEAEDTeeAKHLWEESWDDDDTSDDFSAQLKEELK------
```

B

```
Drosophila_melanogaster       ----MSAPDKE----KEKEKEETNNKSEDLGLLEEDDEFEEFPAEDFRVG
Anopheles_gambiae             ----MS--DKEN---KDKPK-------LDLGLLEEDDEFEEFPAEDWAGN
HUMAN                         ----MS----------EKKQ------PVDLGLLEEDDEFEEFPAEDWAGL
MOUSE                         ----MS----------EKKQ------PVDLGLLEEDDEFEEFPAEDWAGL
Zebrafish                     ----MS----------EKKQ------TVDLGLLEEDDEFEEFPAEDWTGL
Xenopus_laevis                ---MSS----------DKKP------PVDLGLLEEDDEFEEFPTEDWTGF
Neurospora_crassa.            ----MASTQPKNDAKSTEPKPEQPVTEKKTAVLEEDDEFEDFPVDDWEAE
Saccharomyces_cerevisiae      MSTDVAAAQAQSKIDLTKKKNE----EINKKSLEEDDEFEDFPIDTWANG
                                   :          :          .   ********:** : :

Drosophila_melanogaster       ------DDEEELNVWEDNWDDDNVEDDFSQQLKAHLESK--KMET-
Anopheles_gambiae             K-----EDEEELSVWEDNWDDDNVEDDFNQQLRAQLEKH--K----
HUMAN                         ------DEDEDAHVWEDNWDDDNVEDDFSNQLRAELEKHGYKMETS
MOUSE                         ------DEDEDAHVWEDNWDDDNVEDDFSNQLRAELEKHGYKMETS
Zebrafish                     ------DEDEDAHVWEDNWDDDNVEDDFSNQLRAELEKHGYKMETS
Xenopus_laevis                ------DEDEDTHVWEDNWDDDNVEDDFSNQLRAELEKHGYKMETS
Neurospora_crassa.            DTEAAKGNNEAKHLWEESWDDDDTSDDFSAQLKEELKKVEAAKKR-
Saccharomyces_cerevisiae      ETIKS-NAVTQTNIWEENWDDVEVDDDFTNELKAELDRY--KRENQ
                                        :**:.*** :..***. :*: .*.
```

C

```
HUMAN              1   ---------- ---------- ------MSEK KQPVDLGLLE EDDEFEEFPA
MOUSE              1   ---------- ---------- ------MSEK KQPVDLGLLE EDDEFEEFPA
Zebrafish          1   ---------- ---------- ------MSEK KQTVDLGLLE EDDEFEEFPA
Drosophila_m       1   ----MSapDK Ek-------E KEKEET-NNK SE--DLGLLE EDDEFEEFPA
Neurospora_c       1   ----MA--ST QPKNDAKSTE PKPEQpVTEK KTAV----LE EDDEFEDFPV
Xenopus_laev       1   m--------- ---------- -----S-SDK KPPVDLGLLE EDDEFEEFPT
Saccharomyce       1   mstdVA--AA QAQSKIDLTK KKNEEI-NKK S-------LE EDDEFEDFPI
Anopheles_ga       1   ----MS--DK ENKD------ ---------- KPKLDLGLLE EDDEFEEFPA

HUMAN             25   EDWAGLDE-- ----DED-AH VWEDNWDDDN VEDDFSNQLR AELEK----H
MOUSE             25   EDWAGLDE-- ----DED-AH VWEDNWDDDN VEDDFSNQLR AELEK----H
Zebrafish         25   EDWTGLDE-- ----DED-AH VWEDNWDDDN VEDDFSNQLR AELEK----H
Drosophila_m      37   EDFRVGDD-- ----EEE-LN VWEDNWDDDN VEDDFSQQLK AHLES----K
Neurospora_c      41   DDWEAEDtEA AKGNNEA-KH LWEESWDDDD TSDDFSAQLK EELKKveaaK
Xenopus_laev      26   EDWTGFDE-- ----DED-TH VWEDNWDDDN VEDDFSNQLR AELEK----H
Saccharomyce      41   DTWAng--ET IKSNavtqTN IWEENWDDVE VDDDFTNELK AELDR----Y
Anopheles_ga      29   EDWAGNKE-- ----DEEeLS VWEDNWDDDN VEDDFNQQLR AQLEK----H

HUMAN             64   GYKMETS
MOUSE             64   GYKMETS
Zebrafish         64   GYKMETS
Drosophila_m      76   --KMET-
Neurospora_c      90   --Kr---
Xenopus_laev      65   GYKMETS
Saccharomyce      85   --KRENQ
Anopheles_ga      69   --K----
```

Figure 3.12: Multiple alignments of the proteasome DSS1 subunit from different organisms using A) PSI-BLAST, B) ClustalW, and C) DIALIGN. Lower case letters means a part of the sequence that is not significantly aligned.

sertions will be placed differently in the master sequence depending on which other sequence it is aligned with. Another approach is to align all sequences pairwise with all other sequences and establish the difference between every pair. Such a map is called a distance matrix, and from this it is possible to obtain an estimate of which sequences are most related (a cluster), and aligning those first, and then align all the prealigned clusters against each other. This is basically what is implemented in the most used multiple alignment program, ClustalW alias ClustalX [Thompson et al., 1994]. First is calculated a score for the alignment between each pair of the sequences. These scores are then used to calculate phylogenetic tree, or a dendrogram, using the clustering method UPGMA (see Chapter 5). Having calculated the dendrogram, the sequences are aligned in larger and larger groups. Each of these alignments consists of aligning 2 alignments, using profile alignments, which are the alignment of 2 groups of already aligned sequences. The method is an extension of the profile method of Gribskov et al. [1987] for aligning a single sequence with an aligned group of sequences. With a sequence-to-sequence alignment, a weight matrix such as BLOSUM62 is used to obtain a score for a particular substitution between the pairs of aligned residues. In profile alignments, however, each of the two input alignments are treated as a single sequence, but you calculate the score at aligned positions as the average substitution matrix score of all the residues in one alignment vs. all those in the other, e.g., if you have 2 alignments with I and J sequences respectively the score at any position is the average of all the I times J scores of the residues compared separately. Any gaps that are introduced are placed in all of the sequences of an alignment at the same position. However, all gaps in the ends of the sequences are free. This might give some artifacts, especially when sequences of different length are aligned. Newer multiple alignment algorithms implemented in programs such as T-Coffee [Notredame et al., 2000] and DIALIGN [Morgenstern, 1999] handle these problems much better, but the algorithms behind them will not be described in this book. Figure 3.12 is an example of the differences in the results, using different alignment algorithms/programs. Note that PSI-BLAST will only return local alignments, and that the result is based on pairwise alignments to the query sequence, i.e., no clustering has been involved.

3.4 DNA Alignments

Untill now only protein alignments have been described. The basic algorithms and programs used for DNA alignment, however, are the same as for proteins. DNA alignments are much more difficult since at each position, we can have one of only four different bases as opposed to one of twenty in peptide alignments. So we will not have a specific substitution matrix like BLOSUM or PAM

but rather take a step back and use a general substitution score for any match or mismatch but still using affine gap penalties. This makes the probability of any given substitution equally high, and so the significance of the final alignment will be lower. Some nucleotide matrices, however, do have different substitution scores for transitions (Dealing with DNA/RNA sequences from coding regions, however, gives an opportunity to shortcut the alignment by actually aligning the translation products, rather than the actual DNA sequences. This approach has been implemented in most alignment software packages, including FASTA (tfasta [Pearson and Lipman, 1988, Pearson, 1996]) and BLAST (tblast [Altschul et al., 1990, Altschul and Gish, 1996]). In this basic but strong approach, gaps in the aligned DNA sequences will only occur in multiples of triplets. This will, however, not catch examples correctly where frameshifts have actually happened, leading to major changes of larger or smaller parts of the translated protein. For such investigations the programs GenA1 [Hein and Støvlbaek, 1994, 1996] and COMBAT [Pedersen et al., 1998] can be used, but only for pairwise alignments. For multiple alignments an automatic method exists that will translate DNA to peptide, do the multiple alignment using DIALIGN [Morgenstern, 1999], and return the final alignment at the DNA level [Wernersson and Pedersen, 2003]. Multiple DNA alignments are especially useful for investigating the evolution on the molecular level (molecular evolution). With such alignments it is possible to examine exactly which positions in the DNA are more or less likely to undergo mutations that survive and are transferred to the progeny. We can also calculate the chance that a given codon will only allow mutations that will not lead to an amino acid change (silent mutations or synonymous mutations) and compare it to the chance that a substitution leads to an amino acid change (nonsynonymous mutations). This ratio is called dN/dS and an example of such a calculation is given in chapter 7.

3.5 Molecular Evolution and Phylogeny

Phylogenies reveal evolutionary relationships between organisms and specific sequences. In recent years molecular phylogenies have started to play a major role in epidemiological studies of pathogens. These studies provide information about where and when a virulent strain can arise. Not only human pathogens but also viral and bacterial disease-causing agents of livestock are of importance, as such outbreaks can cause great economic loss, as well as increase the chance of a possible cross-species infection. Recent developments of new methods for isolating, amplifying, and sequencing RNA isolated from small samples of blood or tissue have made the molecular phylogeny of pathogens a rapidly expanding research field. Moreover, since many pathogens can mutate at much higher rates than eukaryotes, it is possible to obtain the

phylogeny of sequences that diverged only recently.

One interesting application of molecular phylogeny is represented by analysis of the origins of HIV epidemics. Exactly when simian immunodeficiency virus (SIV) was transmitted from nonhuman primates to humans, giving rise to the human immunodeficiency virus (HIV), is still under investigation. Korber et al. [2000] used a phylogenetic analysis of the viral sequences with a known date of sampling to estimate the year of origin for the main group of HIV viruses (HIV-1 M), the principal cause of acquired immunodeficiency syndrome (AIDS). AIDS is caused by two divergent viruses, HIV-1 and HIV-2. HIV-1 is responsible for the global pandemic, while HIV-2 has, until recently, been restricted to West Africa and appears to be less virulent in its effects. SIV viruses related to HIV have been found in many species of nonhuman primates. By analyzing the molecular divergence of the envelope gene, and applying a model which assumes constant mutation rates through time and across lineages, Korber et al. [2000] estimated that the last common ancestor of the HIV-1 M group appeared in 1931 (with a confidence interval of 1916–1941). Using a different molecular clock analysis, where the mutation rate is allowed to change at splitting events, and also when analyzing a different protein, the same estimates were obtained. This approach only identifies when the common ancestor began to diversify; it does not identify the exact time of transmission. Still, given this estimate, one is able to come up with more precise hypotheses about the transmission event.

3.5.1 Phylogenetic Methods

The starting point of any phylogenetic work is a collection of sequences that might be evolutionarily related. Such a set could be extracted from public databases using some of the tools described previously, or it could be data from one's own work. These sequences must now be aligned by the use of multiple alignment software, such as ClustalW. ClustalW also calculates a distance matrix of your sequences, i.e., the relation of each of your sequences to the other sequences in your alignment. A way to visualize the distances in a distance matrix is a tree-like drawing where the distances along the branches correlates with the distances in the distance matrix. Such a drawing is called a phylogenetic tree. One important point about trees is that they are only useful if the described system has been under vertical evolution (i.e., no horizontal gene transfers and recombination), otherwise a simple tree makes no sense. To calculate the grouping and the branch lengths of such a tree, two major approaches are applicable. One approach is optimization methods that will find the tree that gives the optimal fit to the matrix, e.g., the minimal sum of squared errors. Another approach is clustering methods that is related to the

optimization methods, but is much faster. The clustering methods, however, do not guarantee the optimal solution.

Two major types of trees exist: rooted and unrooted trees. With rooted trees a common ancestor point is used as the origin of the tree, no matter if this is really scientific sane with the given data. In rooted trees the horizontal distance from the leaves to the origin is directly proportional to the amount of changes. Unrooted trees are used to show relations where no common ancestor is given, and only the evolutionary distance between the leaves can be inferred. In both rooted and unrooted trees, the leaves are grouped in clusters. This grouping depends heavily on the algorithm used. Some algorithms just give one of potentially many, more or less equally probable, outputs. Other approaches actually calculate many different solutions and give the most probable outcome with some indication of how reliable a particular solution is.

As a simple example, we will investigate the phylogenetic relationship between HIV and SIV using a set consisting of 27 different gp120 protein sequences from isolates of HIV-1, HIV-2, chimpanzee SIV, and macaque monkey SIV. The gp120 protein of HIV is crucial for binding of the virus particle to target cells. It is the specific affinity of gp120 for the CD4 protein that targets HIV to those cells of the immune system that express CD4 on their surface (e.g., helper T lymphocytes, monocytes, and macrophages). ClustalW is used to align the sequences (figure 3.13) and, as mentioned earlier, ClustalW also clusters the most related sequences. The information from this clustering can subsequently be used to produce a phylogenetic tree (figure 3.14).

The phylogenetic tree from the analysis (see figure 3.14) shows two separate clusters. One contains SIV from chimpanzee (SIVCZ) together with the HIV-1 sequences, while the other contains SIV from macaque/sooty mangabey together with HIV-2. This indicates that HIV-1 originated from one event where the virus was transmitted from (presumably) chimpanzee to human, while HIV-2 originated from a second, independent event where the virus was transmitted from (presumably) macaque to human.

3.6 Viral Evolution and Escape: Sequence Variation

Coexistence of pathogens with their hosts imposes an evolutionary pressure both for the host immune systems and the pathogens. The coexistence depends on a delicate balance between the replication rate of the pathogen and the clearance rate by the host immune response. Throughout the animal and plant kingdoms we see several quite different strategies developed by the host immune systems to defend themselves against intruders. Similarly, the pathogens have developed an array of immune evasion mechanisms to escape their elimination by the host's immune system.

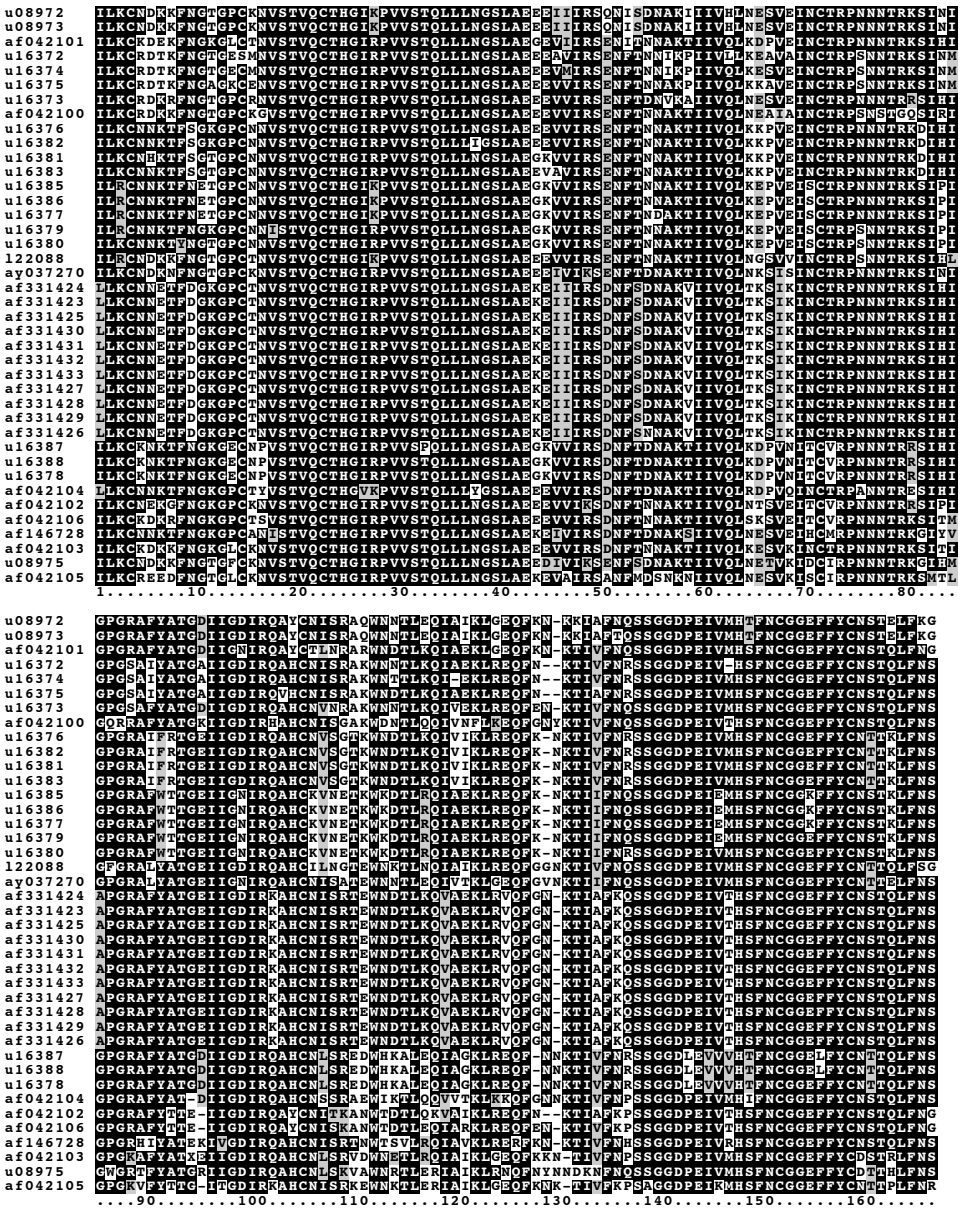


Figure 3.13: ClustalW alignment of 27 HIV/SIV gp120 sequences. The output is modified with the BOXSHADE program.

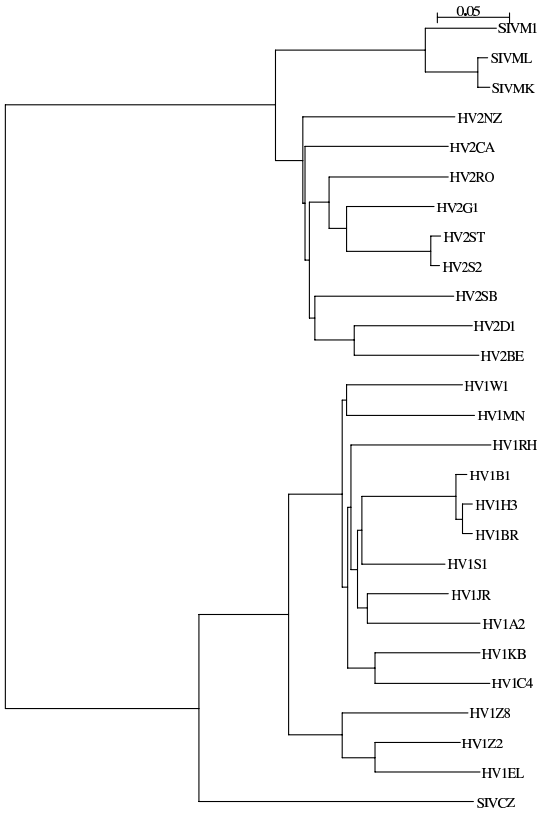


Figure 3.14: A rooted tree of 27 aligned HIV/SIV gp120 sequences. HV1XX=HIV-1 sequences, HV2XX=HIV-2 sequences, SIVMX=SIV (macaque), SIVSX=SIV (sooty mangabey), SIVCZ=SIV (chimpanzee).

We can divide the immune evasion mechanisms (mainly of viruses) broadly into three categories that allow:

1. avoiding the humoral immune response,

2. interfering with the cellular immune response,

3. disrupting the immune effector functions, e.g., by expressing some cytokines.

The humoral response is impaired whenever the antibody binding sites on a protein (often on the surface) mutate in such a way that binding is no longer possible. Especially neutralizing antibodies, i.e., the antibodies that can block infection of the cells by the pathogen, cause a high selection pressure on the virus to mutate. The most straightforward way of identifying such mutants is via sequence analysis of the pathogenic samples. The first step is to align the sequences to pinpoint which regions of the pathogen are mutating. This may be the region that is under the strongest selection pressure by the antibodies. However, it could also be areas with no constrains. Such alignments demonstrate that the most typical examples of escape from antibody response occur in the influenza virus and HIV. The human body can rapidly mount neutralizing antibodies against the major surface protein of the influenza surface protein, hemagglutinin. The influenza virus evades this humoral response by two mechanisms [Gorman et al., 1992]. First, using point mutations, the viral variants can escape neutralization, but this does not cause severe disease, since there will still be some unaltered epitopes that can be recognized. Second, if RNA segments are exchanged between different strains, the hemagglutinin protein can gain a totally different structure. In such a case, the antibodies made during previous infections are no longer functional and severe pandemics can occur [Claas and Osterhaus, 1998]. Interestingly, the phylogenetic analysis of the hemagglutinin protein shows that the antigenic evolution of the influenza virus is punctuated, i.e., some mutants cause epidemics for almost eight consecutive years, while others last only for two or three years [Smith et al., 2004]. Since the 1960s (when the first sequences were collected) every viral mutant has been able to cause an epidemic for at least two years, after which enough individuals will have acquired immunity to limit the spread significantly (herd immunity).

Similarly, the cytotoxic T lymphocyte (CTL) response can be abrogated whenever peptide binding of MHC molecules or binding of the T cell receptor to the MHC-peptide complex is disturbed. It is relatively difficult to observe such escapes, because they are different for each individual, depending on her or his MHC background. Therefore many CTL escape variants can be circulating in a host population without one becoming the dominant mutant. Only in chronic infections like HIV and hepatitis B is it possible to find these escape mutants in a patient. Again, for HIV we have an extensive amount of data to analyze CTL escape mutants. Using sequence analysis it is possible to see that escape mutations are not spread all over the viral genome, because HIV is not able to tolerate changes equally well in all proteins. HIV has very flexible proteins like the envelope protein, gp160, where up to 35% of the sequence can be different from the wild-type virus [Gaschen et al., 2002]. On the other hand, for some proteins, like capsid protein p24, the surface cannot tolerate point mutations without a severe loss of viral fitness [von Schwedler et al., 2003,

Leslie et al., 2004].

An effective vaccine should be able to target the parts of a pathogenic genome that are quite conserved even under the above-mentioned selection pressures. For example, given that less than a 2% amino acid change can cause a failure in cross-reactive immunity of the influenza vaccine [Korber et al., 2001b], it is obvious that for an HIV vaccine to use the envelope protein would be futile. One approach to deal with such large diversity is to use the consensus or the ancestral virus sequence as a vaccine. Such sequences have the advantage of being central and most similar to circulating strains. Another, safer approach would be to design epitope vaccines, which again requires choosing the most conserved epitopes. But the selection of such epitopes also requires computational analysis that goes beyond what simple sequence comparison techniques can handle, as the binding specificities are influenced by correlations between amino acids present at different peptide positions. A solution to this problem is to use machine learning techniques (see chapter 5).

3.7 Prediction of Functional Features of Biological Sequences

During experimental analysis of the immune system, proteins of unknown function are typically being identified as key players using high-throughput gene expression or proteomics data. The functional assignment of such immune system–related proteins also often requires sequence analysis that goes beyond what can be solved by simple sequence alignment methods. In most genomes no more than 40 to 60% of the proteins can be assigned a functional role based on sequence similarity to proteins with known function. Traditionally, protein function has been related directly to the 3D structure of the protein chain of amino acids, which currently, for an arbitrary sequence, is quite hard (in the general case, impossible) to compute. As the sequence, in a given biochemical context, determines the structure, functional information between two sequences can be transferred by comparing the sequence of amino acids by aligning the two against each other. This method is fast and powerful, but only solves part of the problem: it is still impossible to determine that two quite different sequences encode proteins with essentially the same biochemical function.

Several different methods have been developed which do not rely on direct sequence similarity, but on features which go beyond sequence-wide similarity, such as the gene position in the genome, or integration of local or global protein features. One such method, ProtFun, does not, like sequence alignment, compare any two sequences, but operates in the “feature” space of all sequences. ProtFun is therefore complementary to methods based on alignment and the inherent, position-by-position quantification of similarity

between two sequences and their amino acids [Jensen et al., 2002, 2003]. This particular method is still entirely sequence-based and does not require prior knowledge of gene expression, gene fusion, or protein-protein interaction.

For any function assignment method, the ability to correctly predict the functional relationship depends strongly on the function classification scheme used. One would, e.g., not expect that a method based on coregulation of genes will work well for a category like "enzyme," since enzymes and the genes coding for their substrates or substrate transporters often display strong coregulation at the gene and protein levels.

The ProtFun approach to function prediction is based on the fact that a protein is not alone when performing its biological task. It will have to operate using the same cellular machinery for modification and sorting as all the other proteins do. Essential types of post-translational modifications (PTMs) include glycosylation, phosphorylation, and cleavage of N-terminal signal peptides controlling the entry to the secretory pathway, but hundreds of other types of modification exist (a subset of these will be present in any given organism). Many of the PTMs are enabled by local consensus sequence motifs, while others are characterized by more complex patterns of correlation between the amino acids close or far apart in the sequence.

This suggests an alternative approach to function prediction, as one may expect that proteins performing similar functions would share some attributes even though they are not at all related at the global level of amino acid sequence. As several powerful predictive methods for PTMs and localization have been constructed, a function prediction method based on such attributes can be applied to all proteins where the sequence is known.

3.7.1 The ProtFun Method

The ProtFun method integrates (using an artificial neural network approach; see chapter 5 for a general introduction) many individual attribute predictions and calculated sequence statistics (out of many more tested for discriminative value) (see figure 3.15). The integrated method predicts functional categories which can be defined in various ways. The method predicts, e.g., whether a sequence is likely to function as an enzyme, and if so, its category according to the classes defined by the Enzyme Commission. The same scheme can be used to predict any other set of functional classes, including highly specific ones, such as "ligand gated ion channel." It can, for example, be used to identify hormones, growth factors, receptors, and ion channels in the human genome as defined by the Gene Ontology Consortium gene function classification scheme. Obviously, even though such methods produce predictions with false positives and false negatives, they can provide essential clues, e.g., to selecting an assay

if the confidence scores are sufficiently high.

The method uses combinations of attributes as input to the neural network for predicting the functional category of a protein. Combinations of attributes can be selected by evaluating their discriminative value for a specific functional category, say proteins involved in transcription or proteins being transporters. Attributes useful for function prediction must not only correlate well with the functional classification scheme, but must also be predictable from the sequence with reasonable accuracy.

Interestingly, the combinations of attributes selected for a given category also implicitly characterize a particular functional class in an entirely new way. This type of method identifies, without any a priori ranking of their importance, the biological features relevant to a particular type of functionality, say attributes which are discriminative for two different categories of ion channels.

The success of the method indicates that (even predicted) PTMs correlate strongly with the functional categories and this fits well with general biological knowledge. For proteins with "regulatory function" one of the most important features turned out to be phosphorylation, consistent with the fact that reversible phosphorylation is a well-known and widely used regulatory mechanism. Glycosylation was also found to be a strong indicator for regulatory proteins. The most important single feature for distinguishing between enzymes and nonenzymes turned out to be predicted protein secondary structure. This also makes sense, as enzymes are known to be overrepresented among all-alpha proteins where the amino acid chain forms an alpha-helix structure, and more rarely are found to be all-β proteins, where the structure is rich in β-sheet.

3.7.2 Individual Sequence Prediction

The ProtFun method can be used to characterize the entire genome, but it is perhaps best suited for obtaining functional hints for individual sequences for later use in assay selection and design. As an example we can take the human prion sequence which is being associated with the Creutzfeldt-Jacob disease. The functionality of this protein, which seems to produce no phenotype when knocked out in mice, was for a long time not fully understood. The ProtFun method predicts (see figure 3.16) with high confidence that the human prion sequence belongs to the transport and binding category, and also that it is very unlikely to be an enzyme. Indeed, prions have now been shown to be able to bind and transport copper, while no catalytic activity has ever been observed. Interestingly, as the prion is a cell surface glycoprotein (expressed by neural cells) it has a distinct pattern of post-translational modification, which most likely contains information which can be exploited by the prediction method

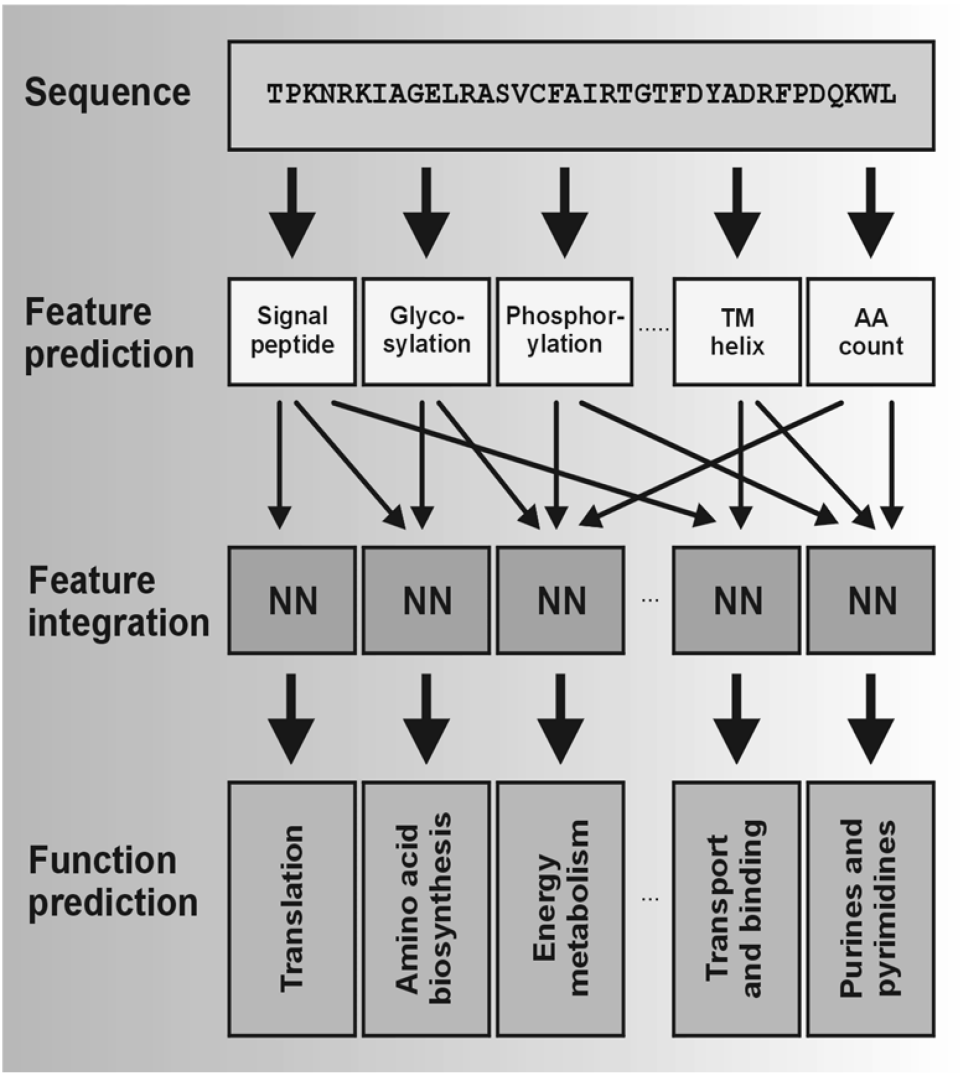


Figure 3.15: The ProtFun neural networks that predict the function of proteins in protein feature space. Each sequence is converted into features and then the networks (NN) integrate these features and provide a prediction for the affinity toward different functional categories. For different categories different protein features will have discriminatory value. During training (using experimentally characterized data) the most discriminative features are determined for each category.

```
######### ProtFun 1.1 predictions ##########

>PRIO_HUMAN

# Functional category                   Prob

  Amino_acid_biosynthesis               0.020
  Biosynthesis_of_cofactors             0.032
  Cell_envelope                         0.146
  Cellular_processes                    0.053
  Central_intermediary_metabolism       0.130
  Energy_metabolism                     0.029
  Fatty_acid_metabolism                 0.017
  Purines_and_pyrimidines               0.528
  Regulatory_functions                  0.013
  Replication_and_transcription         0.020
  Translation                           0.035
  Transport_and_binding              => 0.831

# Enzyme/nonenzyme                      Prob
  Enzyme                                0.250
  Nonenzyme                          => 0.750

# Enzyme class                          Prob
  Oxidoreductase (EC 1.-.-.-)           0.070
  Transferase    (EC 2.-.-.-)           0.031
  Hydrolase      (EC 3.-.-.-)           0.057
  Isomerase      (EC 4.-.-.-)           0.020
  Ligase         (EC 5.-.-.-)           0.010
  Lyase          (EC 6.-.-.-)           0.017
```

Figure 3.16: The prediction output from the ProtFun method for the human prion protein, PRIO_HUMAN. The method produces three types of output for functional categories: broad cellular role, enzyme classes, and Gene Ontology categories, only the two first are included here for reasons of space. The number of Gene Ontology categories predicted is growing and is currently around 75. The numerical output can be used, for example, to select an assay, or the order in which different assays should be selected, when confirming experimentally the function of an uncharacterized protein. The ProtFun method is made available at www.cbs.dtu.dk/services.

for functional inference.

The neural network was not transferring functional information just by identifying by sequence similarity from the nearest neighbor in sequence space used to train the system, as the maximal similarity between the prion sequence and the data set used to train and test the ProtFun method was only 14.8% identity at the amino acid level to a proline-arginine-rich repeat protein. Predictions like these are very useful when resolving protein function, because they can be used to generate specific hypotheses and direct laboratory experiments for sequences where no information at all can be obtained by alignment.

3.7.3 Predicting Functional Categories for Systems Biology: the Cell Cycle as an Example

Characterization of the immune system also requires that genes and proteins are grouped into subsystems, where the biochemical task of each protein may be highly different. The ProtFun method can also be used to group sequences in this manner. As an example with relevance for the immune system, we describe here a version of the method that predicts whether a protein is encoded by a periodically transcribed, cell cycle regulated gene, or not. The ability of a cell to replicate itself is one of the most fundamental features of life, and also of disease, most importantly in relation to cancers. The hundreds of genes maintaining the cell cycle work together in a highly robust manner, making it possible for cells to divide under many different growth conditions and other influences from the environment. The robustness is achieved by sophisticated regulation making the periodic gene expression highly stable. The eukaryotic cell cycle is regulated at many levels, from transcription and translation to posttranslational modification and targeted protein degradation. Proteins need not only be produced, but also be removed again when no longer needed. The cell cycle molecular machinery consists of highly diverse proteins, with little sequence similarity.

A key technique being used to elucidate which genes are involved in a given subsystem is the DNA microarray method (see section 5.1). This is also the case for the cell cycle, where gene expression measurements are made during many different time points of the cycle. Unfortunately, many of the "lists" of genes, which have been produced in this way do not agree as much as expected, even if these studies have produced highly valuable information de Lichtenberg et al. [2003, 2004]. Part of the disagreement relates to differences in experimental conditions and procedures, but a large fraction is presumably related to basic noise problems in the DNA microarray technology when measuring the expression level of weakly expressed genes.

The ProtFun function classification technique described above can be used to predict, in feature space, such systems biology related categories de Lichtenberg et al. [2003]. Not all cell cycle related genes are periodic, but many of the key factors enabling the final formation of protein complexes are. The fact that the method with a reasonable high performance is able to separate such two highly diverse categories, demonstrates that many cell cycle proteins indeed display correlations between their features, which are different from those of other proteins. These features include phosphorylation, glycosylation, stability and/or disposition for targeted degradation, as well as localization in the cell.

In relation to the immune system many other sets of proteins creating a given subsystem may also display feature based similarities that can be ex-

ploited in a prediction approach like ProtFun. One aim is of course to identify novel components involved, but also to discover whether such biochemically diverse proteins share features which can be used to describe the biology behind their functionality.

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

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

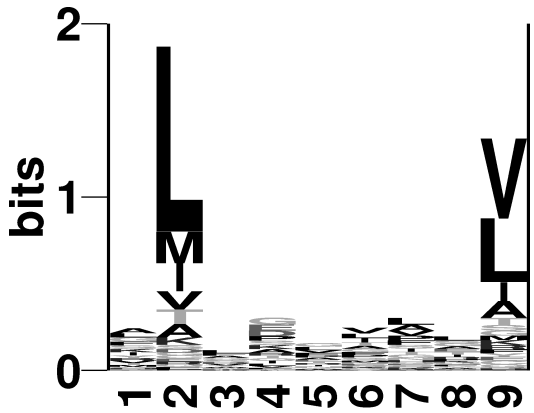


Figure 4.1: Logo showing the bias for peptides binding to the HLA-A*0201 molecule. Positions 2 and 9 have high information content. These are anchor positions that to a high degree determine the binding of a peptide [Rammensee et al., 1999]. See plate 4 for color version.

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.

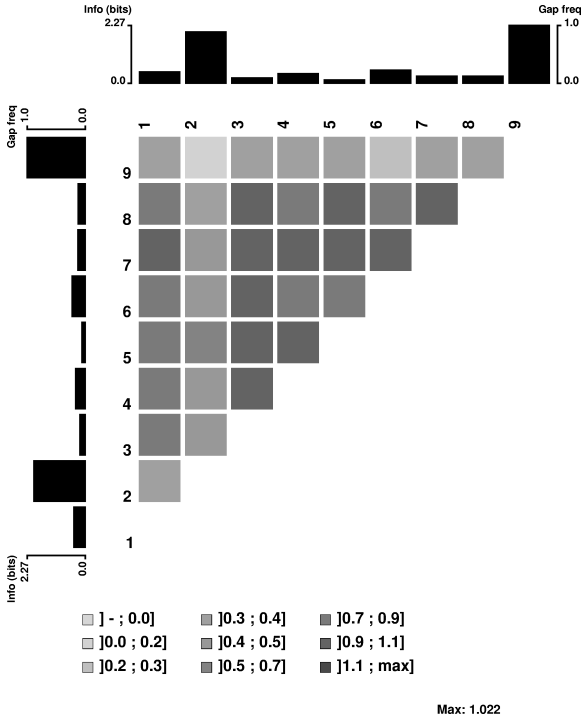


Figure 4.2: Mutual information plot calculated from peptides binding to MHC alleles within the A2 supertype. The plot was made using MatrixPlot [Gorodkin et al., 1999] (http://www.cbs.dtu.dk/services/MatrixPlot/).

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.

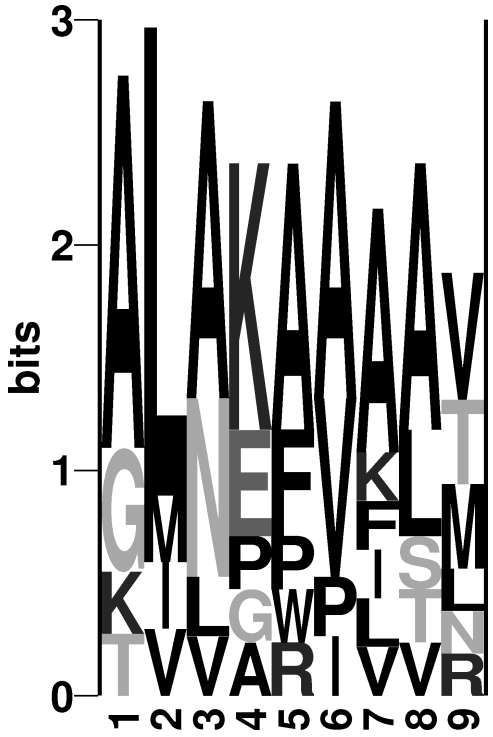


Figure 4.3: Logo representation of the probability matrix calculated from 10 9mer peptides known to bind HLA-A*0201.

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

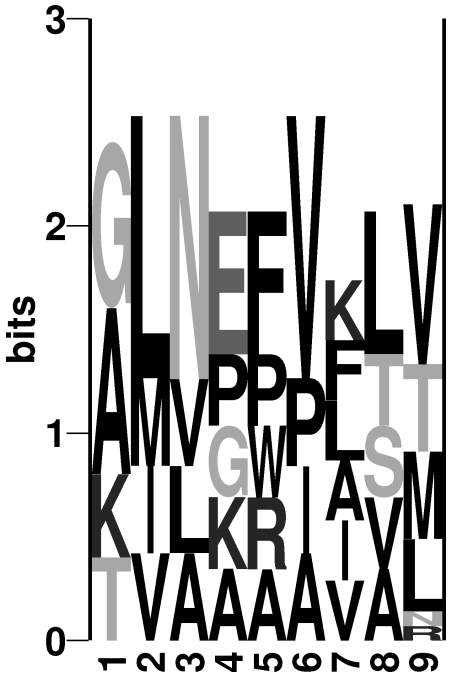


Figure 4.4: Logo representation of the probability matrix calculated from 10 9mer peptides known to bind HLA-A*0201. The probabilities are calculated using the clustering sequence weighting method.

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

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

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, α the effective sequence number minus 1, and β 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 α 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, α 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

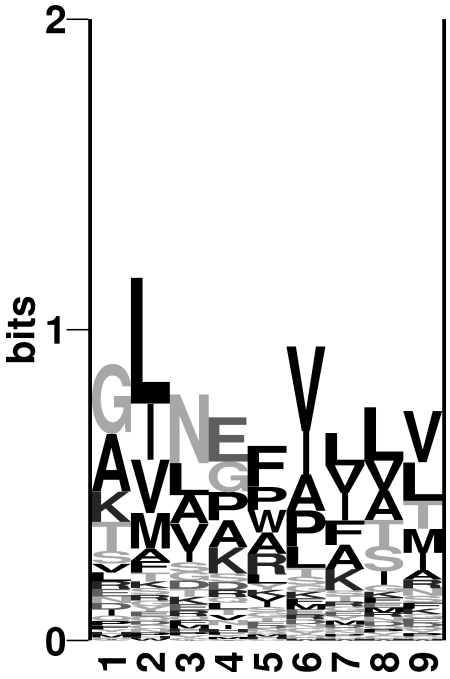


Figure 4.5: Logo representation of the probability matrix calculated from 10 9mer peptides known to bind HLA-A*0201. The probabilities are calculated using both the methods of sequence weighting and pseudocount correction.

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

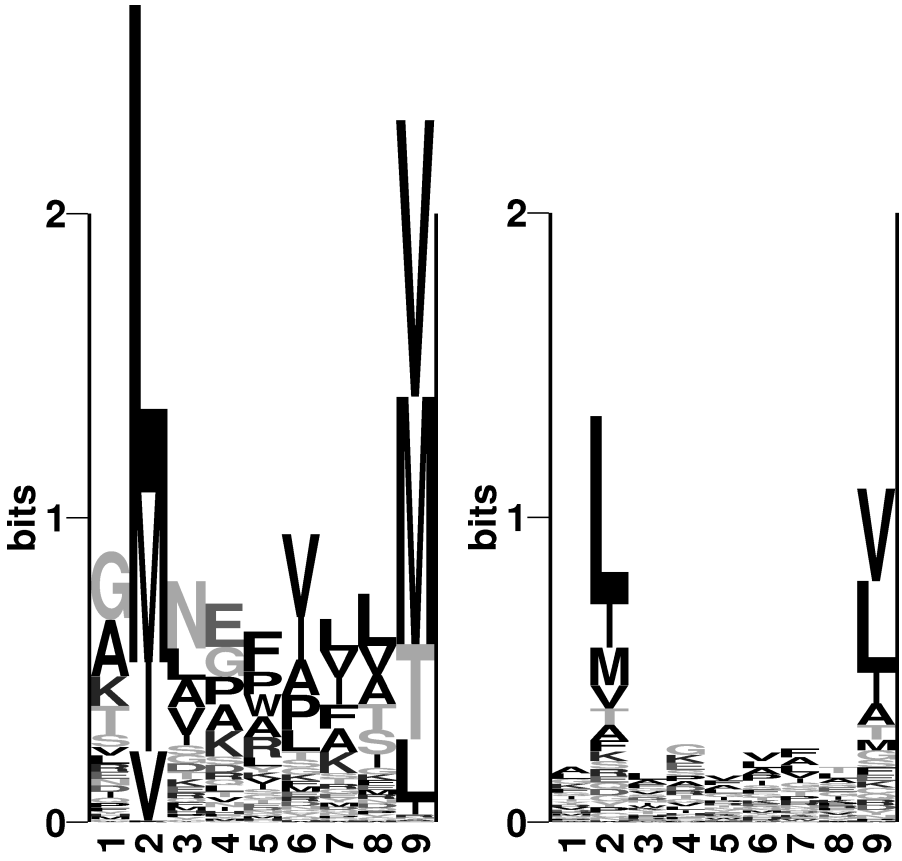


Figure 4.6: Left: Logo representation of the probability matrix calculated from 10 9mer peptides known to bind HLA-A*0201. The probabilities are calculated using the methods of sequence weighting, pseudocount correction, and position-specific weighting. The weight on positions 2 and 9 is 3. Right: Logo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

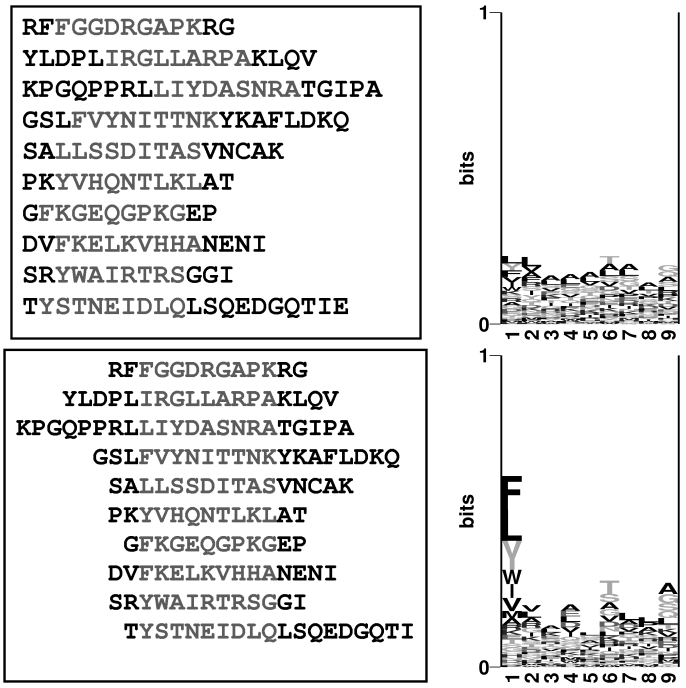


Figure 4.7: Example of an alignment generated by the Gibbs sampler for the DR4(B1*0401) binding motif. The peptides were downloaded from the MHCPEP database [Brusic et al., 1998a]. Top left: Unaligned sequences. Top right: Logo for unaligned sequences. Bottom left: Sequences aligned by Gibbs sampler.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

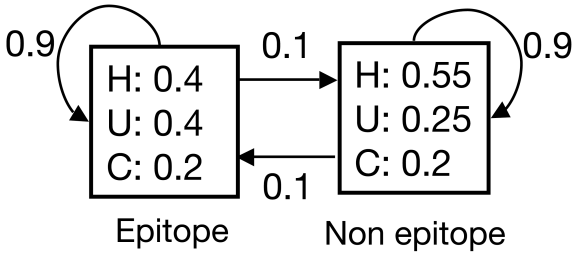


Figure 4.8: B cell epitope model. The model has two states: Epitope E and non epitope N. In each state, three different types of amino acids can be found Hydrophobic (H), uncharged polar (U) and charged (C). The transition probabilities between the two states are given next to the arrows, and the probability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

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

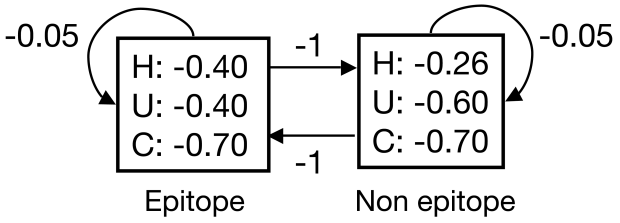
and likewise the value $P_E(H_3)$ is

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

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 π_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

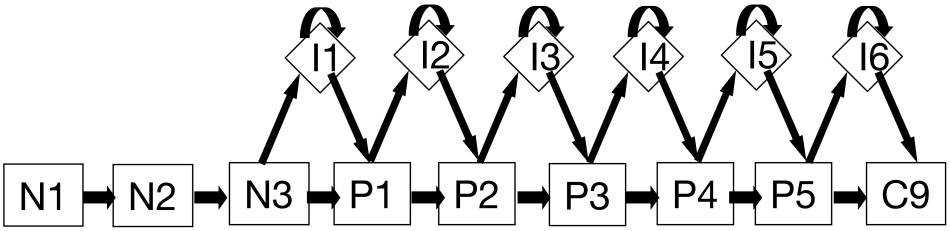


Figure 4.10: HMM for peptide TAP binding. The model can describe binding of peptides of different lengths to the TAP molecules. The binding motif consists of 9 amino acids. The first three N-terminal amino acids, and the last C-terminal amino acids must be part of the binding motif. Each state is associated 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

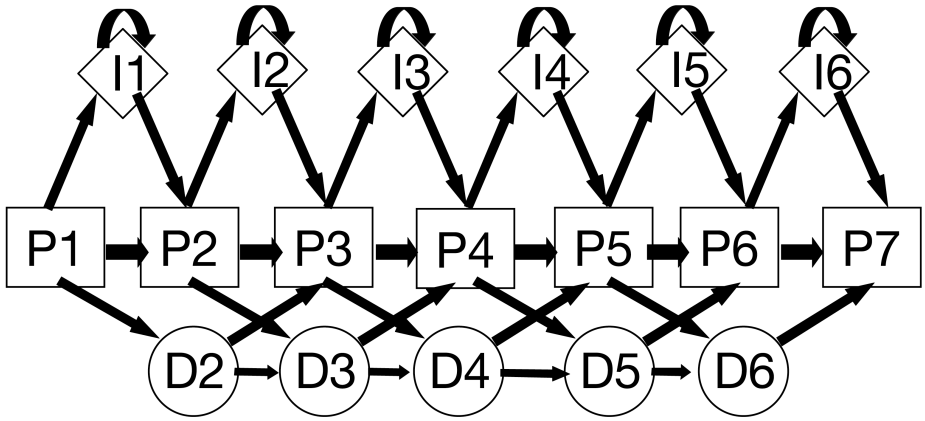


Figure 4.11: Profile HMM with 7 match states. Match states are shown as squares, insertion state as diamonds, and deletion states as circles. Each match and insertion state has an associated probability distribution for matching the 20 different amino acids. Transitions between the different states are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

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

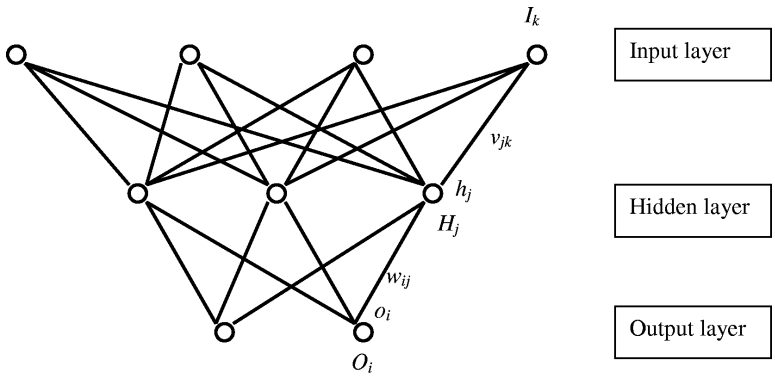


Figure 4.12: Schematic representation of a conventional feedforward neural network used in numerous applications within bioinformatics.

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

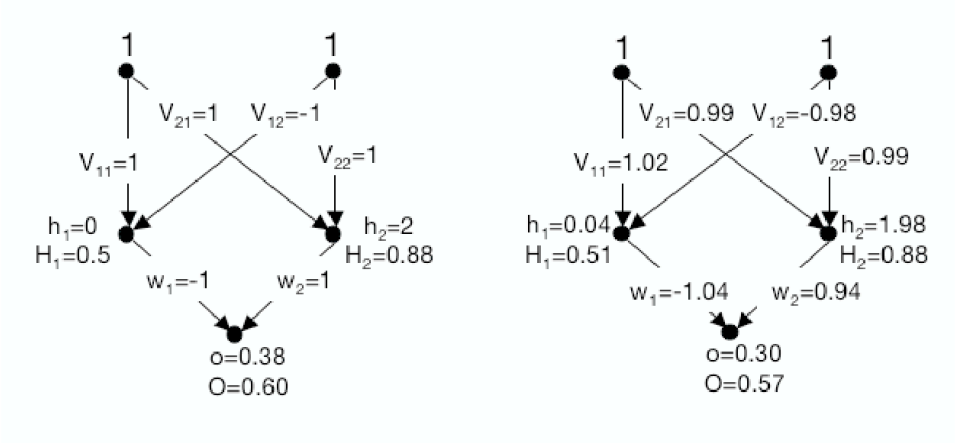


Figure 4.13: Update of weights in a neural network using backpropagation. The figure shows the neural network before updating the weights (left) and the network configuration after one round of backpropagation (right). The learning rate ε in the example is equal to 0.5. Note that this is a large value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

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

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.

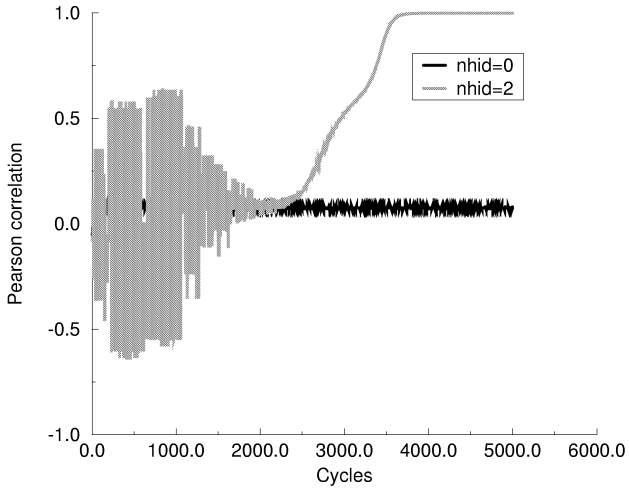


Figure 4.14: Neural network learning curves for nonlinear patterns. The plot shows the Pearson correlation as a function of the number of learning cycles during neural network training. The black curve shows the learning curve for the XOR function for a neural network without hidden neurons, and the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

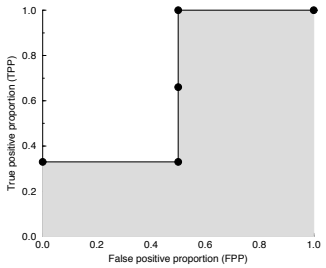


Figure 4.15: Calculation of a ROC curve. The table on the left side of the figure indicates the steps involved in constructing the ROC curve. The pairs of predicted and actual values must first be sorted according to the predicted value. The value in the lower right corner is the A_{ROC} value. In the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