Ph.D. in Electronic and Computer Engineering
Dept. of Electrical and Electronic Engineering
University of Cagliari

Protein Secondary Structure Prediction: Novel Methods and Software Architectures

Filippo Giuseppe Ledda

Advisor: Prof. Giuliano Armano
Curriculum: ING-INF/05

XXIII Cycle
March 2011

Ph.D. in Electronic and Computer Engineering
Dept. of Electrical and Electronic Engineering
University of Cagliari

Protein Secondary Structure Prediction: Novel Methods and Software Architectures

Filippo Giuseppe Ledda

Advisor: Prof. Giuliano Armano
Curriculum: ING-INF/05

XXIII Cycle
March 2011

Acknowledgements

This thesis would not have been possible without the support and the insights of my advisor, prof. Giuliano Armano, and the help of my colleagues of the Intelligent Agents and Soft Computing group. I would like to show my gratitude to Dr. Andrew C.R. Martin for his invaluable advice and collaboration. I am also grateful to the coordinator of the PhD program prof. Alessandro Giua for his efforts towards the quality and international acknowledgment of the program, and to his collaborators Carla Piras and Maria Paola for their helpfulness. Finally, a special thank is dedicated to everyone who has been at my side during these years.

Abstract

Owing to the strict relationship between protein structure and function, the prediction of protein tertiary structure has become one of the most important tasks in recent years. Despite recent advances, building the complete protein tertiary structure is still not a tractable task in most cases; in the absence of a clear homology relationship the problem is often decomposed into smaller sub tasks, including the prediction of the secondary structure. Notwithstanding the large variety of different strategies proposed over the years, secondary structure prediction is still an open problem, and few advances in the field have been made in recent times.

In this thesis, the problem of secondary structure prediction is firstly analyzed, identifying five different information sources related to the biological essence of the problem, in order be exploited in a learning system. After describing a general software architecture and framework aimed at dealing with the issues related to the engineering and set up of prediction systems applied to real-world problems, different techniques based on the encoding and decoding of biological information, together with custom software architectures, are presented.

The different proposals are assessed experimentally. The best improvements are consistent with the recent advances in the field (about 1-2% in the last ten years), confirming the validity of the assumption that the correlation sources identified can be further exploited to improve predictions.

Contents

List of Figures

List of Tables

Glossary and Abbreviations

ANN – **A**rtificial **N**eural **N**etwork. See Section 2.4.
API – **A**pplication **P**rogramming **I**nterface.
BLAST – **B**asic **L**ocal **A**lignment **S**earch **T**ool. See Section 2.3.2.2.
BLOSUM – **Blo**cks of Amino Acid **Su**bstitution **M**atrix. See Section 2.2.1.
BRNN – **B**idirectional **R**ecurrent **N**eural **N**etwork (see refsec:ann,3.5.3)
BP – **B**ack**p**ropagation (see 2.4)
CATH – (Protein structure classification by) **C**lass, **A**rchitecture, **T**opology, **H**omology(see 2.1.4)
DSSP – **D**ictionary of **S**econdary **S**tructure of **P**roteins. See Sections 1.4.3 and 1.4.3.1.
ECOC – **E**rror-**C**orrecting **O**utput **C**odes. See Section 4.4.1.
E-value – **E**xpectation value (see 2.3.2)
GAME – **G**eneric **A**rchitecture based on **M**ultiple **E**xperts. See Chapter 4.
GUI – **G**raphical **U**ser **I**nterface.
HMM – **H**idden **M**arkov **M**odel.
Homology – Evolutionary relationship. See Section 1.2.
MCC – **M**atthews **C**orrelation **C**oefficient. See 3.3.1.
ML – **M**achine **L**earning.
MLP – **M**ulti-**L**ayer **P**erceptron. See Section 2.4.
PCA – **P**rincipal **C**omponent **A**nalysis.
PDB – **P**rotein **D**ata **B**ank. See Section 2.1.3.
Profile – A matrix representing sequential data, which associates a vector to each data element.
PSI-BLAST – **P**osition-**S**pecific **I**terative **BLAST**. See Section 2.3.2.2.
PSSM **P**osition-**S**pecific **S**coring **M**atrix. See Section 2.2.4.
Q_3 – Secondary structure prediction accuracy measure. See section 3.3.1.
Residue – Stands for 'amino acid' within a protein chain. See Section 1.4.1
REST – **Re**presentational **S**tate **T**ransfer
SCOP – **S**tructural **C**lassification **o**f **P**roteins (see 2.1.4)
SM – **S**ubstitution **M**atrix. See Section 2.2.1.
SOV – **S**egment **Ov**erlap. See Section 3.3.1.
SS – (Protein) **S**econdary **S**tructure. See Section 1.4.3.
SSP – **S**econdary **S**tructure **P**rediction. See Chapter 3.
SVM – **S**upport **V**ector **M**achine.
UML – **U**nified **M**odeling **L**anguage.

Introduction

Bioinformatics is a general name given to the application of computer science techniques to biology. It encompasses a lot of different problems and methodologies, whenever the world of living things meets statistics and computation. A more concrete definition relates bioinformatics with the application of computationally intensive techniques to the problems rising from molecular biology, the discipline which studies biological phenomena at molecular level.

The need to store and analyze vast volumes of data, related to recent advances in automatic biological data sequencing, has been the main reason for the birth of a distinct multidisciplinary science. Bioinformaticians have the challenging mission of blending together the knowledge belonging to the vast fields of computer science and biology, in order to use this great amount of data for practical applications in biological studies and medicine.

Research in bioinformatics comprehends many different fields, including genome, sequence, and evolutionary analysis, systems biology, gene expression and structural bioinformatics. In particular, structural bioinformatics is interested in one of the great open problems of biology: the prediction of protein structure from sequence. Since no theoretical models are able to give this relationship are available, techniques coming from the field of machine learning are typically applied to the problem.

Lots of techniques and tools have been proposed in the prediction of the structure of proteins; although good predictions can be obtained when homologues are available, we are still far from obtain good results for ab initio applications. The best advances are limited to restricted sub-problems. The prediction of secondary structure is one, and probably the most studied, of the sub-problems related to protein structure prediction.

Secondary structure prediction is a fascinating field of application for pattern recognition and machine learning techniques. In the course of the development of predictors,

it is important to consider all relevant information sources related to biological aspects of the problem: that requires devising proper statistical analysis to extract the information and suitable software architectures to exploit it.

Structure of the Thesis

This thesis work can be interpreted from two different perspectives. From the purely bioinformatics point of view, which gives the structure and the title to the document, it analyzes the different issues and my contribution around the problem of secondary structure prediction. From a more engineering-oriented perspective, it can be viewed as a compound of different applications of a general architecture and framework I have developed to a "real world" problem.

Chapter 1 is an overview of the general concepts of molecular biology needed to understand the domain of application of the techniques presented in the paper. Starting from the basic *dogma* of molecular biology, which postulates the genesis of proteins from genetic code, the concept of evolution is introduced. The basic components of the cells, home of all the main mechanisms of life, are then described. The final, and more detailed, introductory section is related to proteins and their structure.

Chapter 2 introduces the relevant techniques pertaining to the field of structural bioinformatics. Public sequence and structure databases, which constitute the main source of information of many further analyses, are firstly introduced. Secondly, the fundamental methods for sequence alignment are described. Thirdly, the most important methods and tools for searching in sequence databases are presented. Then, artificial neural networks are introduced; although they cannot be properly considered a method specifically devised for bioinformatics, they are widely used in prediction problems and allow to introduce some of the main issues related to the application of machine learning methods to bioinformatics problems. Finally, the problem of protein structure prediction is introduced, and the main approaches adopted in the field are presented.

Chapter 3 is dedicated to the problem of secondary structure prediction (SSP). The problem is described from a computational point of view and from a historical perspective. The standard issues related to the assessment of the performance of predictors are discussed as well. The problem is then analyzed from a new perspective

which considers the relevant correlations observed in sequences and structures – namely (i) sequence-to-structure, (ii) inter-sequence, (iii) inter-structure (iv) intra-sequence, and (v) intra-structure. Finally, some well known techniques are reinterpreted according to the foregoing analysis.

Chapter 4 presents GAME, a general software architecture and framework which supports the design and the release of predictors. The main insights behind the general architecture are firstly explained. In addition, the implementation of the framework, paying attention to all the details deemed relevant for its practical use, is described. Then, the realization of GAME for secondary structure prediction is described, with most of the details pertaining to the experiments described in the following chapters. Three case studies related to secondary structure prediction are finally presented.

The three final chapters are related to different researches proposed to highlight and exploit the correlations present in the SSP problem, following the philosophy of the GAME architecture. Chapter 5 deals with inter-structure correlation exploring new ways to encode the position-specific information contained in multiple alignments. Chapter 6 presents two different novel researches aimed at exploiting intra-structure correlation with the use of specific output encoding methods together with proper prediction architectures. Chapter 7 presents an innovative approach to the exploitation of inter-structure relationships in the decoding phase of a prediction algorithm.

Appendix A presents a novel method for pairwise sequence alignment in presence of a template, able to take into account the information provided by a beta-contact predictor.

Contribution

I contributed to the field of bioinformatics proposing a novel input encoding method (3), and architectures related to the exploitation of alternative output encoding techniques (4; 5; 6) for secondary structure prediction. In addition, I developed a beta-contact predictor and a pairwise sequence alignment algorithm which uses this predictions in its scoring function (7). The different systems have been realized with the help of a general architecture and framework especially devised to deal with real world prediction problems (8; 9).

Chapter 1

Preliminary Concepts of Molecular Biology

Molecular biology is the discipline which studies biological phenomena at molecular level. In particular, it is interested in the relationship between genes and their functional counterparts, proteins. This relationship is given by the processes of replication, transcription and translation, as stated by the central dogma of molecular biology (Section 1.1). The central dogma, as well as most of the outcomes of molecular biology, are valid for all forms of life, which share the basic mechanisms which permit living things to exist and replicate. This explanation for this great similarity is that all creatures appear to be evolved from a common ancestor (Section 1.2). The basic processes of life happen inside cells (Section 1.3), with the help of specialized proteins and organelles. Many other functions inside cells are carried out with the help of proteins (Section 1.4), complex organic compounds constituted by chains of simpler compounds, the amino acids. A protein's chain composition, commonly referred as primary structure, is determined by the gene which encode for it; the primary structure determines the protein's three-dimensional structure, which in turn determines the protein's function. The relationship between protein chain and structure is the result of a free energy minimization process at molecular level, which cannot be solved nor simulated just with the application of the physics and mathematics determine it. Computational techniques are then usually applied to protein structure prediction and many other problems of molecular biology, giving birth to the discipline of bioinformatics.

1.1 The Central Dogma of Molecular Biology: from genotype to phenotype

The central dogma of molecular biology states that:

- DNA acts as a template to replicate itself,
- DNA is also transcribed into RNA,
- RNA is translated into protein.

In the majority of organisms, DNA constitutes the genetic material, and determines the genotype. It can be seen as the model for potential development and activity of any individual.

DNA molecules are long, linear, chain of nitrogenous bases containing sequential information in a four-letter alphabet. Its structure as a self-complementary double-helix (see Figure 1.1) is the basis of a complex mechanism of self-replication. DNA replication allows inheritance and development of complex organisms. Imperfections in replication are the foundation of evolution.

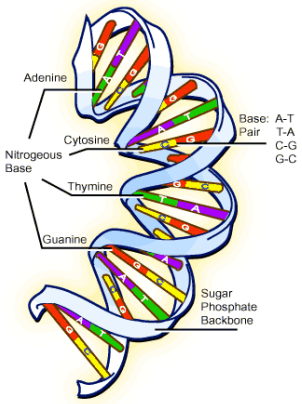


Figure 1.1: The DNA double helix.

The genetic information is implemented through the synthesis of RNA and proteins. However, in most organisms not all of the DNA is expressed as proteins or RNAs. Some regions of the DNA sequence are devoted to control mechanisms, and a substantial amount of the genomes of higher organisms does not appear to have any clear function ('junk' DNA).

Proteins are the molecules responsible for much of the structure and activities of organisms. An exon is a stretch of DNA which translates into protein. An intron is a region between two exons. Cellular machinery recognizes and splices together the proper segments, based on signal sequences within the genome. In the synthesis of proteins, successive triplets of letters (*codons*) from the DNA sequence specify successive amino acids and the boundaries of the protein within the code; stretches of DNA sequences encipher amino acid sequences of proteins. The correspondence between codons and aminoacids is called the 'genetic code'; there is a standard well-known code for all organism which can differ in part for some distant forms of life. Since the possible triplets are $4^3 = 64$ and the number of aminoacids plus start/stop codons are 22, different codons encode the same aminoacid. An exon is transcripted into mRNA, which is then translated into a protein with the help of ribosomes. Figure 1.2 shows the trascription and translation process, which comprehends the synthesis of a protein starting from a fragment of DNA.

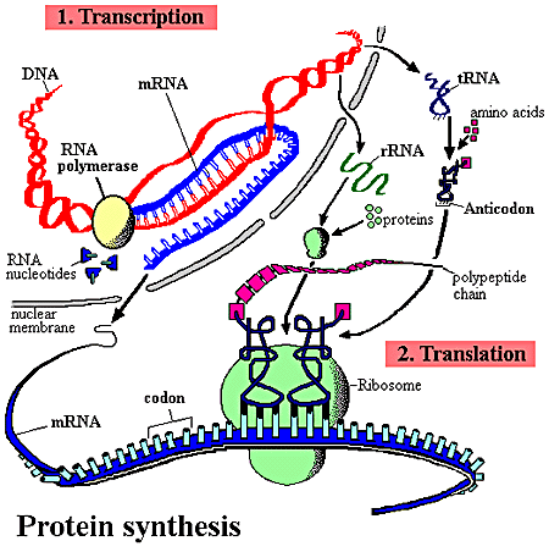


Figure 1.2: The transcription and translation process from the DNA to protein sequence.

1.2 Evolution and Homology

There are so many different kinds of life, and they live in so many different ways. But when we look at the inner mechanisms of life, we can see a great uniformity.

The reason for all this uniformity is that all organisms appear to have evolved from a common ancestor (see Figure 1.3).

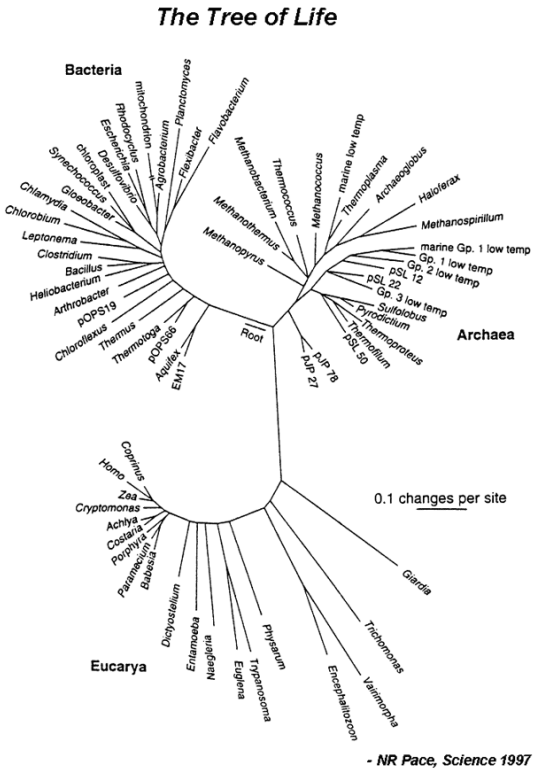


Figure 1.3: Phylogenetic tree of life; all forms of life appear to be generated from a common ancestor.

Evidence of evolution can be seen both looking at the phenotype and genotype. The first phylogenetic trees where constructed just on the basis of anatomical observations, and these relationships have been in great part confirmed by the modern studies on the gene sequences. From a simplified view, the process of evolution can be seen as if, starting from a common ancestor, multiple mutations (substitutions, insertions and deletions of one or more bases) occur, thus generating new genes and proteins. A mutation on a gene affects the sequence of the translated protein where it generates a non-synonymous codon. Although mutations can be considered in principle random, the selective pressure controls it, keeping the mutations from affecting the functionality of genes.

Genes (or proteins) are called *homologous* when there is evidence that they descend

from the same common ancestor. We can also distinguish two main different kinds of homology: *orthology* and *paralogy*. Two genes are orthologous when they belong to different species and originated from a common ancestor (speciation event). Orthologous proteins usually have similar structure and same or similar function. Two genes are paralogous when they are the result of a duplication of a gene in a specie (gene duplication event). Duplicated genes may evolve with different selective pressures, which sometimes lead to a differentiation in function.

Homology can be verified through the comparison of gene sequences, aligned in order to put in evidence the similarities (see Section 2.2). It is worth noting that although measurements such as percent sequence identity or similarity scores in a sequence alignment are usually adopted to verify homology, it is wrong to use expressions such as "percent homology" or "homology score": two genes or proteins can just be homologues or not homologues.

1.3 Cells

Cells are the home of all the main basic life mechanisms. All cells contain cytoplasm and genetic material, are enclosed in a membrane and have the basic mechanisms for translation.

Membrane is the boundary between a cell and the outside world. Membranes are made of phospholipids: the presence of a hydrophobic (repelled by water) lipid with a hydrophilic (attracted to water) phosphate group gives a natural orientation ti the membrane, keeping water and other materials from getting through the membrane, except through special pores or channels.

Nuclei are the defining feature of eucaryotic cells. The nucleus is separated from the rest of the cell by a nuclear membrane.

Cytoplasm is the gel-like collection of substances inside the cell. The cytoplasm contains a wide variety of different substances and structures.

Ribosomes are large molecular complexes, composed of several proteins and RNA molecules. The function of ribosomes is to assemble proteins.

Mitochondria and *chroloplasts* are cellular organelles involved in the production the energy that powers the cell. Mitochondria use oxygen to obtain energy from food. They are in fact distinct organisms which have estabilished a symbiotic relationship with

eucaryote organisms. Chloroplasts are the equivalent of mithocondria for eucaryotic plant cells.

There are other organelles found in eucaryotic cells: the *endoplasmic reticulum* the *Golgi apparatus*, *Lysosomes*. Some cells have other structures, such as *vacuoles* of lipids for storage.

There are more than 200 different specialized cell types in a typical vertebrate. Yet despite all of this variation, all of the cells in a multicellular organism have exactly the same genetic code. Gene expression determines whether or not the product a gene codes for is produced, and how much is produced in a cell, so determining its peculiarity.

1.4 Proteins

Proteins are the primary components of living things, and carry out most of the cell functions. Proteins provide structural support and the infrastructure that holds a creature together; they are enzymes that make the chemical reactions necessary for life possible; they are the switches that control gene expression; they are the sensors that see and taste and smell, and the effectors that make muscles move; they are the detectors that distinguish self from nonself and create an immune response. Most of the proteins are 'globular', in the sense that they assume a globe-like shape in their natural water environment. An important class of non-globular proteins are membrane proteins, which shape depends on the interaction with cell membrane.

Proteins are complex molecules composed by linear sequences of smaller molecules called amino acids. There are twenty naturally occurring amino acids. Protein chains may contain from some dozens to thousands amino acids assembled by a peptide bond. Peptide bonds occur between the nitrogen atom at the end of one amino acid and the carbon atom at the carboxyl end of another. The portion of the original amino acid molecule integrated into the protein is often called a *residue*.

1.4.1 Amino Acids

There are 20 naturally synthesized amino acids. Each amino acid shares a basic structure, consisting of a central carbon atom (C), an amino group (NH3) at one end, a carboxyl group (COOH) at the other, and a variable sidechain (R), as shown in Figure 1.4. The composition of the side chain determines the shape, the mass, the volume and

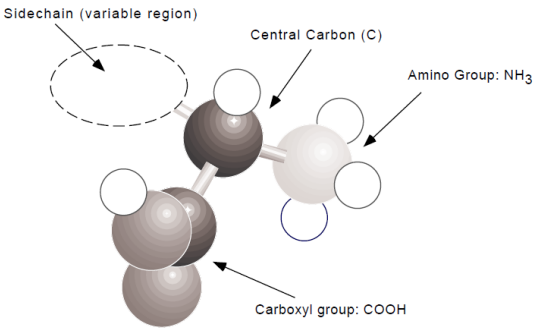


Figure 1.4: The general structure of an amino acid.

the chemical properties of an amino acid. According to the properties of the side chain amino acids are classified as:

- *Polar/non-polar*: the degree that its electrons are distributed asymmetrically. A non-polar molecule has a relatively even distribution of charge. Some polar aminoacids are positively or negatively charged in solution.
- *Hydrophobic/Hydrophilic*: hydrophobic residues tend to come together to form compact core that exclude water. Because the environment inside cells is aqueous (primarily water), these hydrophobic residues will tend to be on the inside of a protein, rather than on its surface.
- *Aromatic*: an aromatic amino acid forms closed rings of carbon atoms with alternating double bonds (like the simple molecule benzene)
- *Aliphatic*: aliphatic amino acids side chain contains only carbon or hydrogen atoms

Figure 1.5 shows a representation of the amino acids and their properties. Amino acids nomenclature have been standardized by IUPAC, and comprehends four alternatives (Table 1.1).

1.4.2 Primary Structure

The sequence of amino acid residues that form a protein is called the protein's *primary structure*. Similarly as the DNA, the primary structure can be represented as a sequence using the 1-letter IUPAC nomenclature for amino acids. More general representation

Trivial name	**3-letter**	**1-letter**	**Systematic name**
Alanine	Ala	A	2-Aminopropanoic acid
Arginine	Arg	R	2-Amino-5-guanidinopentanoic acid
Asparagine	Asn	N	2-Amino-3-carbamoylpropanoic acid
Aspartic acid	Asp	D	2-Aminobutanedioic acid acid
Cysteine	Cys	C	2-Amino-3-mercaptopropanoic acid
Glutamine	Gln	Q	2-Amino-4-carbamoylbutanoic acid
Glutamic acid	Glu	E	2-Amino-3-carbamoylpropanoic acid
Glycine	Gly	G	Aminoethanoic acid
Histidine	His	H	2-Amino-3-(1H-imidazol-4-yl)-propanoic acid
Isoleucine	Ile	I	2-Amino-3-methylpentanoic acid
Leucine	Leu	L	2-Amino-4-methylpentanoic acid
Lysine	Lys	K	2,6-Diaminohexanoic acid
Methionine	Met	M	2-Amino-4-(methylthio)butanoic acid
Phenylalanine	Phe	F	2-Amino-3-phenylpropanoic acid
Proline	Pro	P	Pyrrolidine-2-carboxylic acid
Serine	Ser	S	2-Amino-3-hydroxypropanoic acid
Threonine	Thr	T	2-Amino-3-hydroxybutanoic acid
Tryptophan	Trp	W	2-Amino-3-(lH-indol-3-yl)-propanoic acid
Tyrosine	Tyr	Y	2-Amino-3-(4-hydroxyphenyl)-propanoic acid
Valine	Val	V	2-Amino-3-methylbutanoic acid
Unspecified	Xaa	X	-

Table 1.1: The IUPAC nomenclature for amino acids.

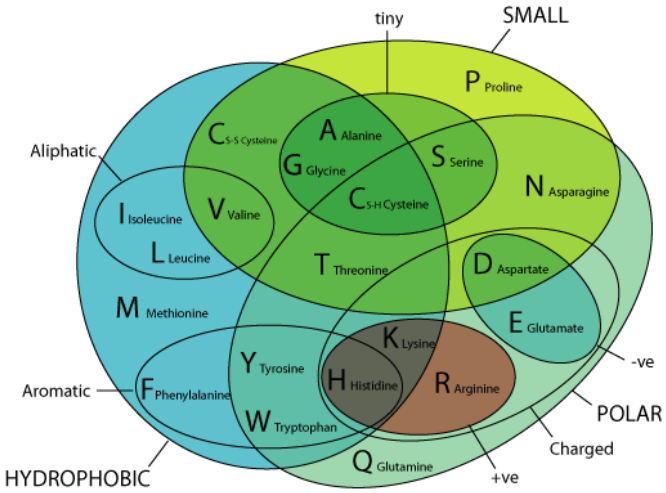


Figure 1.5: Amino acid properties (Taylor's Venn diagram (1)).

of the primary structure is given by *profiles*: in a general meaning, we can see a profile as a matrix which associates a vector to each amino acid of a protein (see Section 5). Issues about the representation of a primary structure are further discussed in Section 3.4.2.

1.4.3 Secondary Structure

Secondary structure refers to local conformations of amino acid residues that are seen repeatedly in proteins. Secondary structures are stabilized by hydrogen bonds (relatively weak bonds between an electro-negative atom and a hydrogen). There are two main kinds of secondary structure:

α-helices are corkscrew-shaped conformations where the amino acids are packed tightly together.

β-sheets (also called β-pleated sheets) are made up of two or more adjacent strands of the molecule, extended so that the amino acids are stretched out as far from each other as they can be. Each extended chain is called a β-strand. Two or more β-strands are held together by hydrogen bonds to form a β-sheet. There are also two main categories β-sheet: if strands run in the same direction we have a parallel β-sheet. If they run in the opposite direction we have an anti-parallel β-sheet.

Other kinds of secondary structure have been defined: The 3_{10}-*helix* and π-*helix*, are less common helix patterns. Strands formed by isolated residues are also called β-

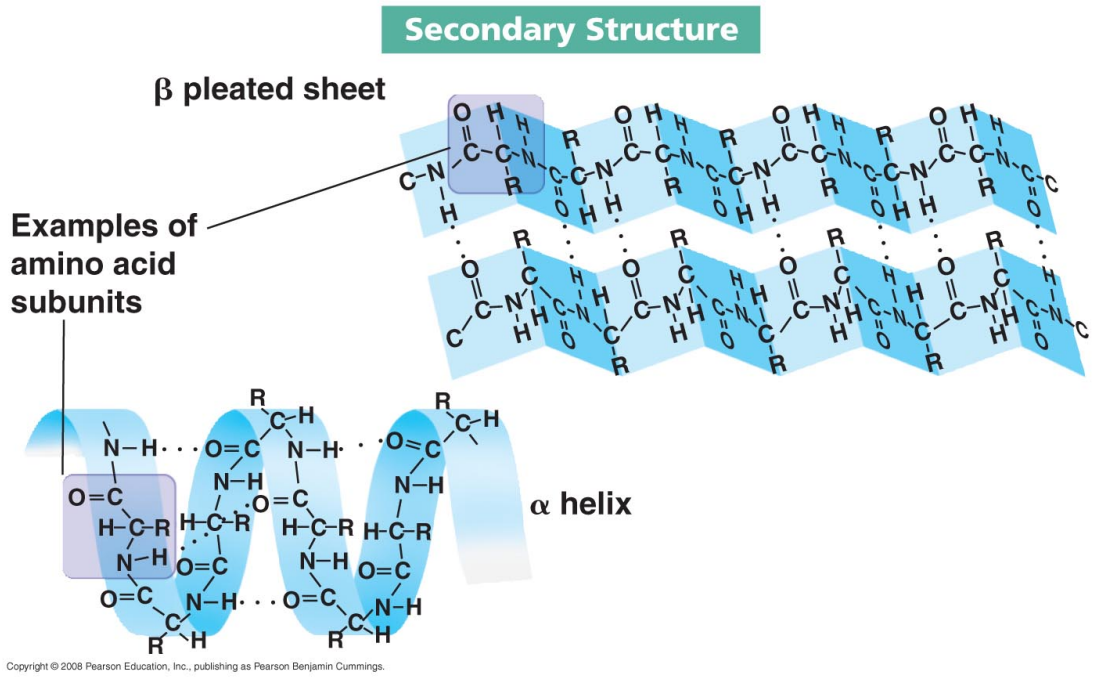


Figure 1.6: An example of spatial arrangements of amino acid backbones occurring in α-helices and β-sheets. Dotted lines indicate the hydrogen bonds.

bridges. Tight *turns* and loose, flexible *loops* link the more 'regular' secondary structure elements. The conformations that are not associated with a regular secondary structure are called *random loops* or *coils*.

The secondary structure of a protein can thus be represented by the sequence of annotations for each of its residues. The Dictionary of Secondary Structure of Proteins (DSSP) is the most widely used set of annotations for secondary structures. The output categories of DSSP are H, G, I, E, B, T, S, and C (the latter is actually a "none" assignment represented by white space). In SSP, a simplified version of $DSSP$, say $DSSP_{HEC}$, is typically adopted, which maps each of the eight initial categories to alpha-helix (H), beta-strand (E), or coil (C). $DSSP_{HEC}$ is related to DSSP throughout a look-up table. The most acknowledged correspondence is reported below:

$DSSP$	H	G	I	E	B	T	S	C
$DSSP_{HEC}$	H	H	H	E	E	C	C	C

1.4.3.1 Automatic Secondary Structure Assignment

Secondary structure is usually obtained by an automatic assignment from the tertiary structure coordinates. The most commonly used programs are DSSP (10) and STRIDE (11). DSSP performs its sheet and helix assignments on the basis of the detection of backbone-backbone hydrogen bonds. An alpha-helix assignment (DSSP state 'H') starts when two consecutive amino acids have (i, i+4) hydrogen bonds, and ends likewise with two consecutive (i-4, i) hydrogen bonds. This definition is also used for 3_{10}-helices (state 'G' with (i, i+3) hydrogen bonds) and for π-helices (state 'I' with (i, i+5) hydrogen bonds) as well. A minimal size helix is set to have two consecutive hydrogen bonds in the helix, leaving out single helix hydrogen bonds, which are assigned as turns (state 'T'). *beta*-sheet residues (state 'E') are defined as either having two hydrogen bonds in the sheet, or being surrounded by two hydrogen bonds in the sheet. This implies three sheet residue types: anti-parallel and parallel with two hydrogen bonds or surrounded by hydrogen bonds. Isolated residues fulfilling this hydrogen bond criterion are labelled as β-bridge (state 'B'). The remaining two DSSP states 'S' and " (space) indicate a bend in the chain and unassigned/other, respectively.

STRIDE uses an empirically derived hydrogen bond energy and phi-psi torsion angle criteria to assign secondary structure, and it is reported to be more similar to annotations provided by expert christallographers than DSSP.

Both DSSP and STRIDE are freely distributed programs, and both output a plain text file with the annotations and additional details. Public data banks collecting the output for the secondary structures obtained for all the PDB are available(for example, at the EBI ftp service).

It must be noticed that there is some grade of uncertainty in the assignment of secondary structures: crystallographers often disagree in their assignment, and the mobility of proteins does not help having a unambiguous assignment. The automatic programs greatly depend on the thresholds used; DSSP and STRIDE agree in 96% of all residues with 64% of the disagreements related to the helix assignment for the same PDB file. For an in-depth examination about secondary structure assignment, see (12).

1.4.4 Tertiary Structure

Proteins fold up to form particular three dimensional shapes, which give them their specific chemical functionality. The link between amino acids provided by the peptide bond has two degrees of rotational freedom, the ϕ and ψ angles. The conformation of a protein backbone (i.e. its shape when folded) can be described as a series of ϕ/ψ angles, using the Cartesian coordinates of the central backbone atom (the alpha carbon, written Ca), or using various other representational schemes. The position of the atoms in a folded protein is called its *tertiary structure* (Figure 1.7).

Figure 1.7: A common representation of tertiary structure.

In a protein's structure can be usually identified one or more *active sites*, which are directly related to its function. Some proteins bind to other proteins or groups of atoms that are required for them to function: for example, the heme group permits hemoglobin to bind oxygen. Often, several structural *domains*, i.e. parts of the protein that can evolve, function, and exist independently of the rest of the protein chain, can be also identified. Moreover, protein structures are not static: they can move and flex in constrained ways, and that can have a significant role in their biochemical function.

1.4.4.1 From Sequence to Structure: the Levinthal's Paradox

The tertiary structure of a protein and, therefore, its biological activity and function, is determined by its amino acid sequence (13). Exactly how the properties of the amino acids in the primary structure of a protein interact to determine the protein's ultimate conformation remains unknown. Although the features of amino acids play some role in protein folding, there are few absolute rules. The conformation a protein finally assumes

will minimize the total 'free' energy of the molecule. For example, packing several large side chains near each other increases the local free energy, but may reduce the energy elsewhere in the molecule. According to the experiments performed by Levinthal (14), the folding process has on average 3^{300} degrees of freedom, which generates a number of alternatives still intractable in computer simulations. This enormous difference between the actual speed of the folding process and the computational complexity for evaluating the corresponding model is also called Levinthal's paradox.

Molecular simulators[1] that use some heuristics for reducing the search space have been developed, but the uncertainty about the degree of approximation of the actual structure limits their use only to very short chains or small perturbations around a known structure.

Given the limits of molecular simulators, in most cases a protein structure must be determined experimentally, or, where possible, with the help of predictors.

1.4.4.2 Experimental Determination of Tertiary Structure

The majority of protein structures are solved with the experimental technique of X-ray crystallography, which typically provides data of high resolution but provides no time-dependent information on the protein's conformational flexibility. A second common way of solving protein structures uses NMR, which provides somewhat lower-resolution data in general and is limited to relatively small proteins, but can provide time-dependent information about the motion of a protein in solution. More is known about the tertiary structural features of soluble globular proteins than about membrane proteins because the latter class is extremely difficult to study using these methods.

1.4.5 Supersecondary Structure

Rao and Rossmann (16) observed structural motifs comprising a few alpha-helices or beta-strands which were frequently repeated within structures. They call them "supersecondary structures" being intermediate to secondary and tertiary structure and suggested that these structures might be due to evolutionary convergence. A variety of recurring structures were subsequently recognised, such as the "Helix-loop-helix" and the "Greek key" (Figure 1.8). Some of these structural motifs can be associated with a

[1] For a general survey about molecular simulators, see (15).

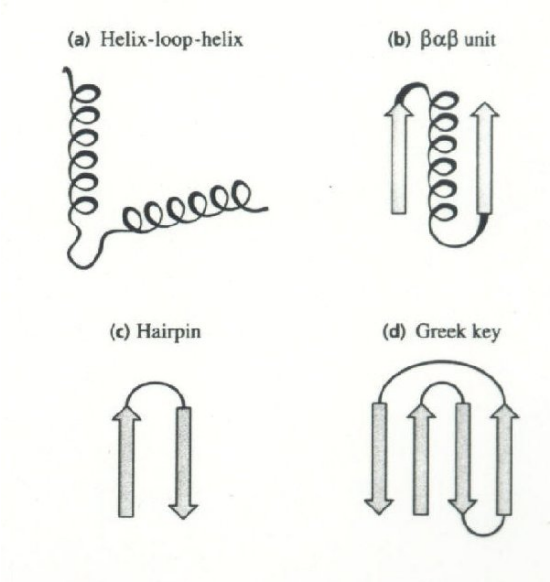


Figure 1.8: Some common structural motifs.

particular function while others have no specific biological function alone but are part of larger structural and functional assemblies.

1.4.6 Quaternary Structure

Quaternary structure is an active conformation of multiple protein chains in one larger complex. A chain may bond with copies of itself or with other proteins to cooperate. Examples of proteins with quaternary structure include hemoglobin (see Figure 1.9), DNA polymerase, and ion channels.

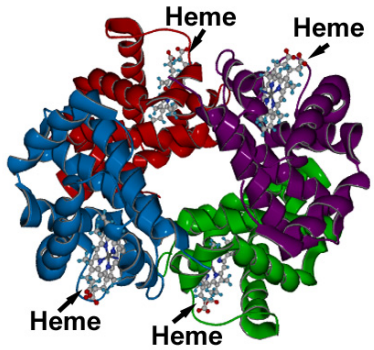


Figure 1.9: The quaternary structure of hemoglobin.

Chapter 2

Resources and Methods for Bioinformatics

Development in the field of bioinformatics is obtained by the sharing of knowledge and the development and use of software tools based on methodologies from lots of different fields related to statistics and computation (see Figure 2.1). Research in bioinformatics encompasses lots of different tasks, which can be clustered into two main aspects:

- *functional:* representation, storage, and distribution of data;
- *analytical:* developing tools to discover new knowledge in data.

The functional aspect is related to the sharing of knowledge about biological data, usually coming from experiments run in laboratories all over the world. This knowledge is shared in public databases and data banks (Section 2.1), maintained by specific public institutions or universities. The adaptation of standard computer science techniques for storage and querying is not trivial: a lot of challenges arise, related to the big amount of data and the distributed and varied nature of the data which is submitted. For example, the same protein structure may be studied by different teams in different experiments run in different times by different laboratories equipped with different machineries with different resolution: the result would be distinct and in either cases imperfect, and one team may not know of the experiment of the other, so giving unrelated identifiers to their structures. In addition, the definition of what is "same" and what is "different" may relevantly change the sense of a query. We may say that two proteins with identical sequences are same, but this would even exclude proteins

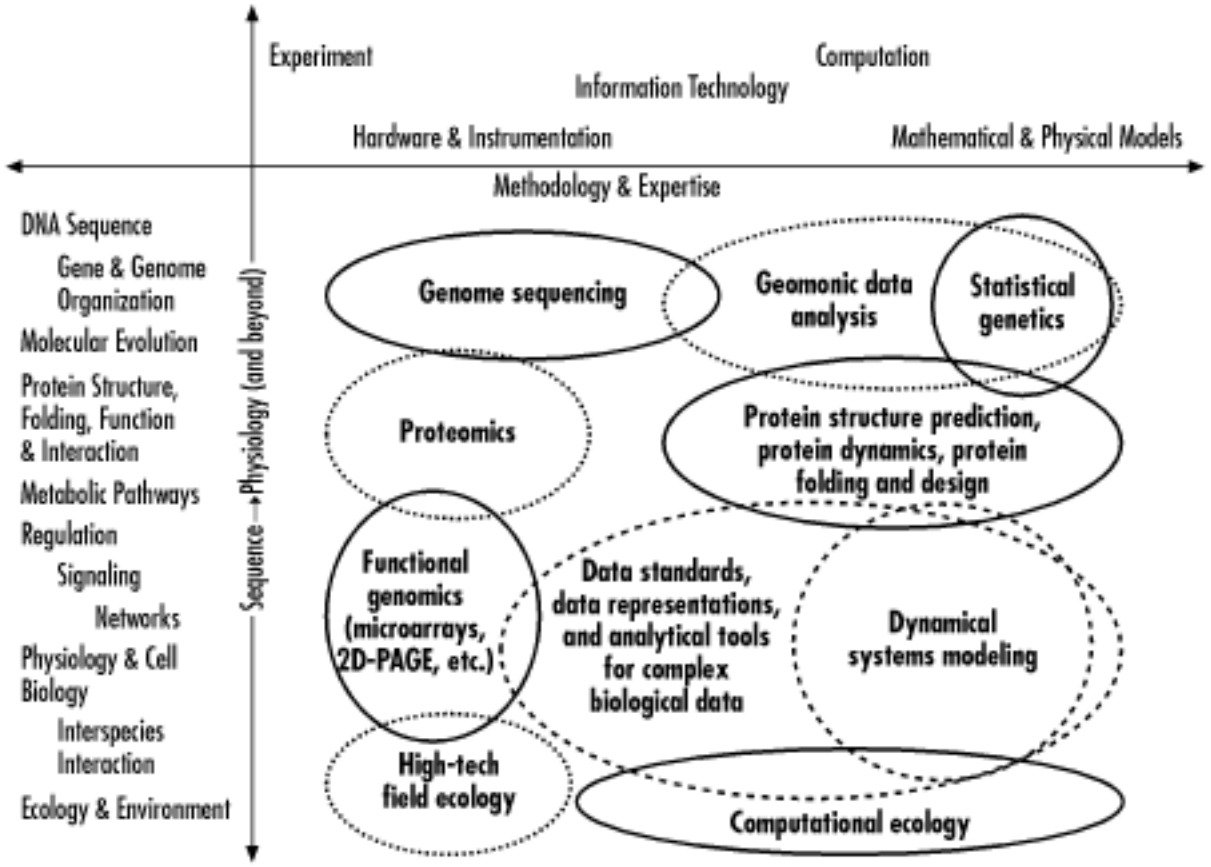


Figure 2.1: Biology and technology (from (2))

from the same specie, which can also be affected by some mutation. The definition of "same" and "different" depends in fact on what we are interested in. Measures of similarity are then needed, and taking into account the evolution process is essential to relate the measure to the biological relationship. Scoring matrices (Section 2.2.1) and sequence alignment techniques (Section 2.2) are used together to perform relevant biological comparison of protein and nucleotide sequences. Owing to the computational limits of sequence alignments, heuristic techniques, such as FASTA and BLAST (Section 2.3.2), are generally adopted while searching for similarities in big sequence databases. BLAST is actually the most used tool by bioinformaticians, and the paper presenting it is one of the most cited in all fields of science (33,867 citations according to Google Scholar!).

The analytical aspect of bioinformatics, which can be considered the more purely "scientific", develops computational techniques mainly based on statistics and artificial intelligence. The large amount of available data is analyzed, filtered, transformed and integrated, with the purpose of discovering the sense beyond this data. In this process, one of the great open problems is the prediction of phenotype from genotype. Phenotype (structural) data is indeed much harder (and expensive) to determine experimentally than genotype (sequential) data, and the knowledge of both is needed for advanced applications: structures are directly related with functionality, while sequences are easier to compare and synthesize.

Understanding the implication between the genetic code and life could have virtually infinite applications, from ad-hoc drug synthesis to the creation of forms of life with the desired characteristics, but we are still far from that. Research in this direction is still at its first steps, and one of the biggest open problems today is the prediction of protein structures (Section 2.5). Since theoretical models able to relate sequence and structure are still not available, comparative techniques and machine learning systems constitute nowadays the best alternatives.

2.1 Public Databases

2.1.1 DNA Sequence Databases

Sequencing DNA has become a routine task in the molecular biology laboratory. Since the first automatic sequencer invented by Leroy Hood in the 1980, lots of advances have been made: the first whole human genome was sequenced in 2003 and we already have companies which offer personal genome sequencing.

Investigators every day submit newly obtained sequences from every form of life to public databases, such as the National Center for Biotechnology Information (NCBI), which manages GenBank[1] (17); the DNA Databank of Japan (DDBJ) [2]; or the European Molecular Biology Laboratory (EMBL)/EBI Nucleotide Sequence Database[3]. GenBank, EMBL, and DDBJ have now formed the International Nucleotide Sequence Database Collaboration [4], which acts to facilitate exchange of data on a daily basis.

[1] `http://www.ncbi.nlm.nih.gov`.

[2] `http://www.ddbj.nig.ac.jp`.

[3] `http://www.embl-heidelberg.de`.

[4] `http://www.ncbi.nlm.nih.gov/collab`.

The accuracy of the sequences collected in the databases can vary mainly depending on the laboratory which performed the sequencing. Also, the databases are by themselves redundant, in the sense that no effort is made to limit the submission of the same sequence multiple times. A non-redundant (*nr*) database is provided at the NCBI as a synthesis of Gen- Bank, EMBL and DDBJ databases.

2.1.2 Protein Sequence Databases

A variety of protein sequence databases exist, ranging from simple sequence repositories, which store data with little or no manual intervention in the creation of the records, to expertly curated databases in which the original sequence data are enhanced by the manual addition of further information in each sequence record.

UniProtKB/Swiss-Prot is a curated protein sequence database providing a high level of annotation (such as the description of the function of a protein, its domains structure, post-translational modifications, variants, etc.). *UniProtKB/TrEMBL* is an automatically annotated database, derived from EMBL and complementary to swiss-prot. The UniProt Reference Clusters (*UniRef*) databases provide clustered sets of sequences from the UniProtKB and selected UniProt Archive records to obtain complete coverage of sequence space at several resolutions while hiding redundant sequences. For instance, *UniRef90* comprehends protein sequences with a maximum sequence identity of 90% one each other. Also the NCBI provides a non-redundant database of proteins (nr).

2.1.3 Protein Structure Databases: The Protein Data Bank

Searching in public databases is the first thing that a researcher in structural bioinformatics usually does while looking for a protein structure. The most important of these databases is the Protein Data Bank (PDB) (18), which collects X-ray crystallography or NMR spectroscopy structures of proteins and nucleic acids submitted by laboratories from around the world. As shown in Figure 2.2 The number of structures stored in the PDB has grown exponentially in the last 20 years.

The contents, stored as plain text files (in the 'PDB' format) are freely accessible and can be downloaded via the websites of its member organizations. The PDB format specifies a set of mandatory and optional records, in which along with information about the experiment and the compounds analyzed, the coordinates of every visible

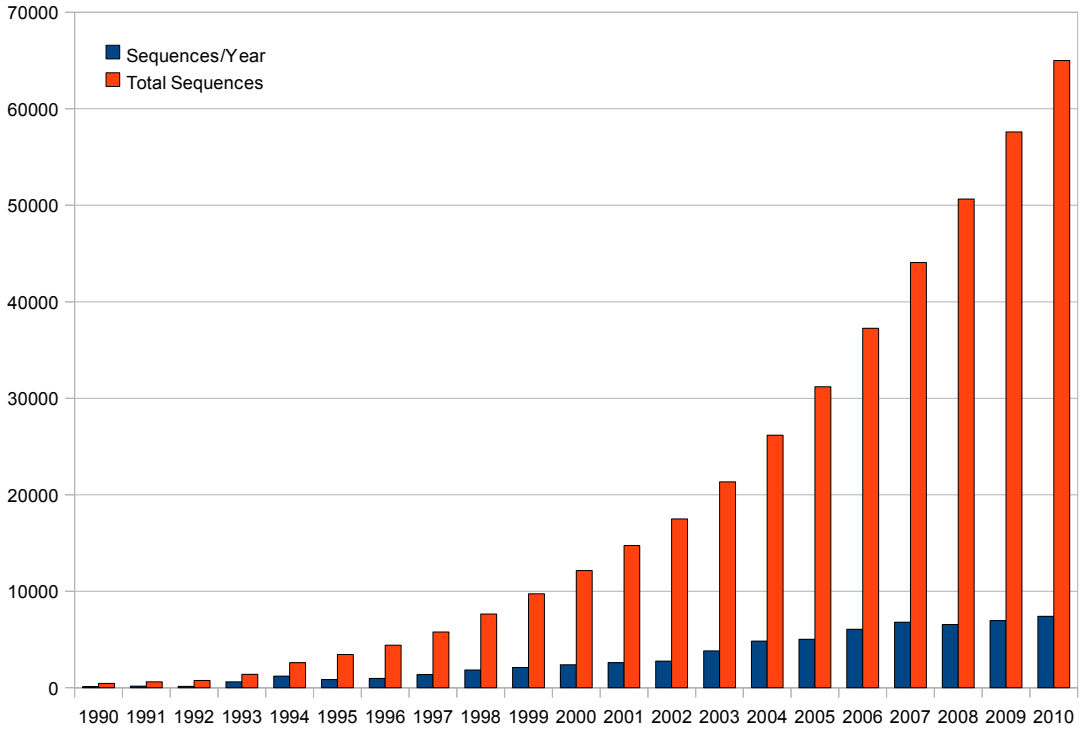


Figure 2.2: Number of protein structures on the PDB since 1990.

atom are given. Many different programs allow to open, visualize and analyze PDB files, such as RASMOL(19) and PYMOL(20).

2.1.4 Protein Structure Classification

Protein structures can be classified according to their folding. The most known classification resources are CATH (21) and SCOP (22).

In CATH proteins are chopped into structural domains and assigned into a hierarchical classification of homologous superfamilies. The classification uses a main four-level structural annotation (*class*, *architecture*, *topology*, *homology*), and five finer levels of sequence identity within a family. For example, at the third level a domain may be of class α/β, *Sandwich* architecture, β-*lactamase* topology (see Figure 2.3). CATH uses a combination of automated and manual techniques which include computational algorithms, empirical and statistical evidence, literature review and expert analysis.

SCOP proposes a classification based on full proteins instead of domains. The hier-

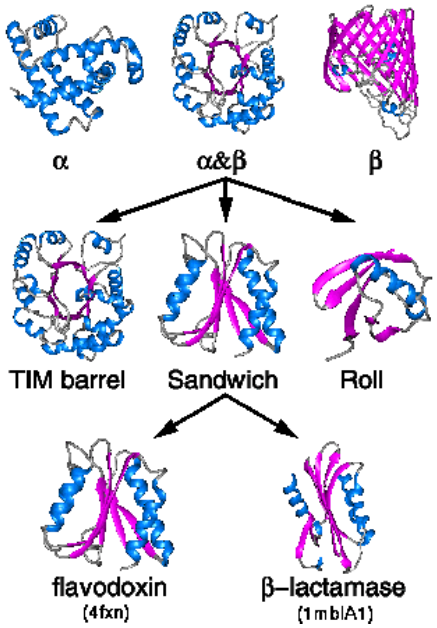


Figure 2.3: A sample CATH hierarchy (class, architecture, topology levels).

archy proposed by SCOP also has four levels: *class*, *fold*, *superfamily*, *family*. Although similar in principles with the CATH's levels, the different definition of each level results in a quite distinct hierarchy. The SCOP classification is mostly based on human expertise.

2.2 Sequence Alignment

Sequence alignment is a fundamental concept in bioinformatics, as it is the base of many further analyses. Since the nucleic acid fragments and protein primary structures data is represented with sequences, related each other by the process of evolution, the alignment of such sequences is necessary to compare genotypes.

The conceptual rule of sequence alignment algorithms is to revert the evolution process in order to put in evidence any conservations and substitutions occurred in the evolution process. In addition, sequence alignment can highlight similarities due to structural or functional reasons. Two sequences are aligned by writing them in two rows:

```
QVQLQESG-AEVMKPGASVKISCKATG---YTFSTYWIEWVKQRPGHGLEWIGEILPGSGSTYYNEKFKG-K
VLMTQTPLSLPVSLGDQASISCKSSQSIVHSSGNTYFEWYLQKPG----QSPKLL----IYKVSNRFSGVPD
```

Characters (i.e. nucleic acids or amino acid residues) placed in the same column represent conservations and substitutions. Gaps (character '-') represent either insertions or deletions.

An alignment algorithm is defined by two components: (i) the score function i.e. a scoring scheme used to evaluate an alignment; (ii) the alignment strategy, i.e. a strategy which gives the succession of substitutions, insertions and deletions which maximize the score.

A scoring scheme's aim is to favour those substitutions, insertions and deletions which have been selected by evolution. Since a lot of factors may participate in the evolutionary selection, finding the correct scoring scheme is still an open problem, which greatly affects the reliability of sequence alignments, especially for distantly related proteins. The most commonly used scoring schemes rely on substitution matrices, which are based on the statistical observation of the substitutions occurred in sets of homologous proteins. In addition to substitution matrices, usually two different costs for gap creation and extension are part of a scoring system, in order to reproduce the fact that a new insertion/deletion is less probable to occur than a yet existing one to be extended (*affine gap penalties*).

Alignment strategies may have a *global* or *local* scope, and different strategies apply when a couple of sequences (*pairwise alignments*) or sets of related sequences (*multiple alignments*) are considered. Global strategies attempt to align every residue, and are suitable when the sequences are similar and of roughly equal size. Local alignments are more useful for dissimilar sequences that are suspected to contain regions of similarity within their larger sequence context.

2.2.1 Substitution Matrices

The PAM (Point Accepted Mutation) matrices, or Dayhoff Matrices (23), are calculated by observing the substitutions in closely related proteins. The PAM1 matrix estimates what rate of substitution would be expected if 1% of the amino acids had changed. The PAM1 matrix is used as the basis for calculating other PAM matrices by assuming that repeated mutations would follow the same pattern as those in the PAM1 matrix, and multiple substitutions can occur at the same site. Using this logic, Dayhoff derived matrices as high as PAM250. Usually the PAM30 and the PAM70 are used.

BLOSUM matrices (24) are based on the analysis of substitutions observed within conserved blocks of aligned protein sequences belonging to different protein families. Like PAM matrices, BLOSUM matrices can be derived at different evolutionary distances. The measure of evolutionary distance is the percentage identity of the aligned blocks in which the substitutions are counted. For instance, the widely used BLOSUM62 matrix is based on aligned blocks of proteins where on average 62% of the amino acid substitutions are identical. This is approximately equivalent to PAM150.

In order to permit a simple use in scoring systems, substitution matrices are normally given in log-odd format: the odds are used instead of probabilities to take into account the different occurrence of amino acids. The use of logarithms is a convenience to obtain the score of an alignment from the sum of the scores for the single substitutions.

Many matrices alternative to PAM and BLOSUM have been proposed; for a review and assessment of substitution matrices, see (25).

2.2.2 Pairwise Sequence Alignment

A pairwise alignment can be represented by a path in matrix, in which one of the sequences represents the rows and the other the columns. A crosswise movement represents a substitution and horizontal and vertical movements represent gaps. The dot-matrix sequence comparison is a technique which permits to visually detect the regions of similarities between two sequences.

The Needleman and Wunsch algorithm (26) is the standard technique for global pairwise sequence alignment. It is based on dynamic programming, an optimization technique which relies on the fact that the solution for the whole problem can be obtained by the solutions for its subproblems. The Needleman and Wunsch algorithm finds the path in the matrix, by storing the optimal score for each cell incrementally.

The Smith and Waterman algorithm(27) is the reference local alignment method and it is also based on dynamic programming. The main difference with the Needleman and Wunsch algorithm is the adoption a breaking threshold when the score for the alignment goes below a certain score while building the matrix.

Both the Needleman and Wunsch and the Smith and Waterman algorithm are suited to be used with substitution matrices and affine gap costs.

2.2.3 Multiple Sequence Alignment

Multiple alignments can be used to evidence conserved regions in a set of proteins. They are often used to perform phylogenetic analyses on protein families.

Dynamic programming techniques can be extended for multiple sequences, but they are not usually purely adopted for sequence alignments with more than a handful of sequences, due to the explosion of the memory and time cost, which grows exponentially with the number of sequences.

Progressive techniques are an effective compromise, and can be scaled to deal with great numbers of sequences. The main idea behind progressive techniques is to start from the most related sequences then and add the other sequence one at a time. The most famous progressive method are CLUSTALW (28) and MUSCLE (29). T-Coffee (30) is another progressive method which also allows to combine results obtained with several alignment methods.

Multiple alignments are often expressed as Hidden Markov Models or Position-Specific Scoring Matrices to describe protein families or motifs.

Figure 2.4: A multiple sequence alignment.

2.2.4 Position-Specific Scoring Matrices

Analysis of multiple sequence alignments for conserved blocks of sequence leads to production of the position-specific scoring matrix, or PSSM. PSSMs are used to search a motif in a sequence, to extend similarity searches, or as sequence encoding for prediction algorithms. A PSSM can be constructed by a direct logarithmic transformation of a

matrix giving the frequency of each amino acid in the sequence. If the number of aligned residues in a particular position is large and reasonably diverse, the sequences represent a good statistical sampling of all sequences that are ever likely to be found. For example, if a given column in 20 sequences has only isoleucine, it is not very likely that a different amino acid will be found in other sequences with that motif because the residue is probably important for function. In contrast, another column from the same 20 sequences may have several amino acids represented by few residues. Hence, if the data set is small, unless the motif has almost identical amino acids in each column, the column frequencies in the PSSM may not be highly representative of all other occurrences of the motif.

The estimates of the amino acid frequencies can be improved by adding extra amino acid counts, called pseudocounts, to obtain a more reasonable distribution of amino acid frequencies in the column (31):

$$p_{ca} = \frac{n_{ca} \cdot b_{ca}}{N_c \cdot B_c} \tag{2.1}$$

where n_{ca} and b_{ca} are the real counts and pseudo-counts, respectively, of amino acid a in column c, N_c and B_c are the total number of real counts and pseudo-counts, respectively, in the column.

Knowing how many counts to add is not trivial; relatively few pseudo-counts should be added when there is a good sampling of sequences, and more should be added when the data are more sparse. Substitution matrices (e.g. BLOSUM or PAM) provide information on amino acid variation. Then, b_{ca} may be estimated from the total number of pseudo-counts in the column by:

$$b_{ca} = B_c \cdot Q_i \tag{2.2}$$

where $Q_i = \sum_i q_{ia}$. q_{ia} is the frequency of substitution of amino acid i for amino acid a in the substitution matrix.

2.3 Searching in Sequence Databases

2.3.1 Database queries

Web pages or services allow queries to be made of the major sequence databases. For instance, a program called ENTREZ (http://www.ncbi.nlm.nih.gov/Entrez) is available

at NCBI. ENTREZ accesses information such as the sequence included with the sequence entry, accession or index number, name and alternative names for the sequence, names of relevant genes, types of regulatory sequences, the source organism, references, and known mutations, thus allowing rapid searches of entire sequence databases for matches to one or more specified search terms. These programs also can locate similar sequences (called 'neighbors' by ENTREZ) on the basis of previous similarity comparisons.

2.3.2 Similarity Searches

Sequence similarity searching is a crucial step in analyzing newly determined protein sequences. A common reason for performing a database search with a query sequence is to find a related gene in another organism; large databases are scanned to identify statistically significant matches.

Similarity can be in principle assessed by scoring sequence alignments between the query sequence and the sequences in a database. However, pairwise similarities (especially if confined to very short regions) can also reflect convergent evolution or simply coincidental resemblance. Hence, percent identity or alignment score should not be used as a primary criterion for homology; matches which have a low probability of occurrence by chance are more correctly interpreted as likely to indicate homology instead.

The *p-value* indicates the probability to encounter a given score (or higher) by chance. The *e-value* (expectation value), is conceptually similar to the p-value: it estimates the probability to find a particular hit score or higher by chance by searching in the current database. For example, should a hit have an e-value of 0.02, there is a one in fifty chance that an alignment of the same or better quality would occur by chance alone. E-value is usually the choice to evaluate similarity searches.

Given the big size of sequence databases, heuristic algorithms for sequence comparison like FASTA (32) or BLAST (33) are usually preferred to dynamic programming algorithms. PSI-BLAST (34) is an iterative version of BLAST which uses PSSMs to find remote homologues. Other tools use HMMs instead of PSSMs to perform similarity searches (35).

Similarity search programs are nowadays the most widely used class of tools by bioinformaticians, and are available to use from the main bioinformatics portals.

2.3.2.1 FASTA

FASTA provides a rapid way to find short stretches of similar sequence between a query sequence and any sequence in a database. Each sequence is broken down into short words a few sequence characters long, and these words are organized into a table indicating where they are in the sequence. If one or more words are present in both sequences, and especially if several words can be joined, the sequences must be similar in those regions.

2.3.2.2 BLAST and PSI-BLAST

BLAST (Basic Local Alignment Search Tool) was developed as a new way to perform a sequence similarity search by an algorithm that is faster than FASTA while being as sensitive. Like FASTA, the BLAST algorithm increases the speed of sequence alignment by searching first for common words or k-tuples in the query sequence and each database sequence. Whereas FASTA searches for all possible words of the same length, BLAST confines the search to the words that are the most significant. For proteins, significance is determined by evaluating these word matches using log odds scores in substitution matrix (typically, BLOSUM62). For the BLAST algorithm, the word length is fixed by default at 3 for proteins. The words considered as significant are then merged, expanded, and finally aligned with the Smith-Waterman algorithm.

PSI-BLAST iteratively applies BLAST for a number of iterations (usually set to 2-3). The first iteration is a normal BLAST run. After each iteration, a PSSM is extracted from the multiple alignment previously generated, and it used to score the next iteration instead of a simple scoring matrix. The use if a PSSM makes PSI-BLAST more sensitive to remote homologues compared to BLAST.

BLAST and PSI-BLAST can be used from a web interface or downloaded for personal use from the NCBI site. A web interface is also provided by the EBI.

2.4 Artificial Neural Networks

Artificial Neural networks (ANNs) are adaptive statistical models based on an analogy with the structure of the brain. ANNs are made of basic units (the *neurons*), which propagate signals to the other connected units. ANNs can be viewed as general input-output relationship estimators; the estimation is achieved after a learning process from

a set of examples, commonly referred as *training set*. For this reason, ANNs are in fact machine learning algorithms and are commonly used in classification, regression and clustering. This section briefly describes the characteristics of neural networks, with particular emphasis on the training of multi-layer perceptron, which is widely recalled in the following chapters. The interested reader could find further examination in Abdi *et al.* (36), Bishop (37), Duda *et al.* (38).

2.4.1 The base units: connections, weights, activation function

A neuron (see Figure 2.5) propagates information to other neurons by weighted connections called *synapses*.

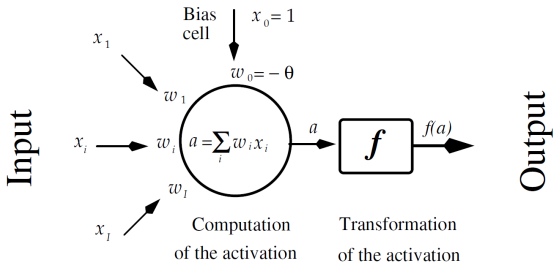


Figure 2.5: A neural network unit.

Information may be provided by other units or by external sources (for input neurons). Synaptic *weights* multiply (i.e., amplify or attenuate) the input information. An *activation function*, which operates on the sum of the input signals (the *activation*), determines the output of the neuron.

Formally, if each input is denoted x_i, and each weight w_i, then the activation is equal to $a_j = \sum x_i \cdot w_{ij}$, and the output y_j is obtained as $y_j = f(a_j)$. Any function whose domain is the real numbers can be used as a transfer function. The most used activation function is the *logistic sigmoid*:

$$sigm(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

The logistic sigmoid has the useful properties of mapping the real numbers into the interval [-1 + 1] and having a derivative which is easily computed, which is useful for the learning process.

2.4.2 Network Architectures

ANNs usually organize their units into several layers. The first layer is called the input layer, the last one is the output layer. The intermediate layers (if any) are called the hidden layers. The information to be analyzed is fed to the neurons of the first layer and then propagated to from layer to layer until the output layer , from which the output is taken. The pattern of connectivity of the neurons determines the *architecture* of a network. The behaviour of a network is completely specified by its architecture, the activation functions and the synaptic weights.

A simple and very popular architecture is the *multi-layer perceptron* (MLP). MLPs are feed-forward neural networks with multiple layers, and can be represented with a directed acyclic graph, in which every neuron of the i_{th} layer is connected (only) to the neurons of the $(i+1)_{th}$ layer (see Figure 2.6).

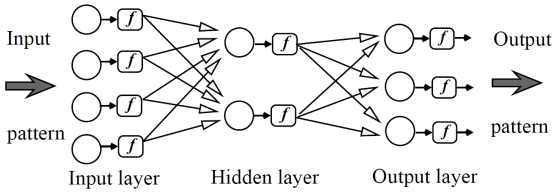


Figure 2.6: A multi-layer perceptron.

MLPs are very popular because they can be now reliably trained, and they can be theoretically used with any regression or classification problem. In particular, with sigmoid activation functions a 3 layer net can approximate any multivariate function relating the input to the output to arbitrary accuracy(39). A limit of MLPs is that they are constructed to deal with static data: they expect a vector of fixed length (the length being the number of the neurons in the input layer) as input, and the length of the output vector is also fixed by the number of output neurons. Still, meta-architectures with sliding windows can permit to use MLPs to deal with dynamic data (such as signals).

Many other neural networks architectures have been proposed in the literature, including recurrent neural networks (40; 41), self-organizing maps (42).

2.4.3 Training Process

The training of a neural network is achieved by modifying the connection weights between its units. In statistical terms, connection weights are parameters to be estimated.

The values of the parameters (i.e., the synaptic weights) can change iteratively as a function of the performance obtained by the network. These changes are made according to learning rules which can be supervised (when a desired output is known and used to compute an error signal) or unsupervised (when no such error signal is used). The *Widrow-Hoff* (also known as *gradient descent* or *Delta rule*) is the most widely known supervised learning rule:

$$\Delta w_{ji} = \eta(t_j - y_j)f'(a_j)x_i. \tag{2.4}$$

The difference between the actual output y_j of the neuron and the desired output t_j are here used as an error signal for units in the output layer[1]. η is a scalar parameter called *learning rate*.

The error signal for the hidden layers is estimated as a function of the error of the units in the following layer. This adaptation of the Widrow-Hoff learning rule is known as error backpropagation (BP, (43)). The BP algorithm was originally developed to be used with MLPs and has been successfully adapted to other network architectures. Training with BP is usually implemented by repeating the backpropagation steps for all the examples in the training set. The examples are resubmitted iteratively, until a stop condition is reached. Simple stop conditions can be a threshold to the error on the training set, or to the number of iterations. The learning rate is a crucial parameter: high values of η may speed up the training, but may also forbid a fine tuning around the minimum. For this reason, a dynamic value of η is often preferred, which decreases during the iterations so having bigger values in the first iterations when the solution is plausibly far and vice-versa. A common adaptation of the delta rule includes a *momentum* term, which adds inertia to the gradient in order to avoid oscillations around local minima.

[1]The given formula is valid for the commonly used square-error function $\sum_j \frac{1}{2}(t_j - y_j)^2$. Other forms of the delta-rule apply for different error functions.

2.4.4 Design of MLP systems

The proper design of a system based on MLPs is usually a manual and iterative process, which requires intuition and expertise in order to obtain the best performance. In particular, a ANN must learn as much as possible from the training examples while avoiding "overfitting". Overfitting is a well-known phenomenon which occurs when learning too much details about training examples causes to lose generalization capabilities (i.e. fitting on unknown data). One reason for overfitting is that training data can be affected by measurement or labeling errors. This is common in real-world problems, and it is particularly evident with biological data. Another interpretation of overfitting refers to the Occam's Razor principle: overfitting occurs when the proposed solution is too complicated for the given problem. That explains why the phenomenon occurs also when the training data is not affected by measurement errors.

The dimension of the hidden layer N_h is an important parameter to consider when designing a MLP. There is no general rule which tells us the best dimension for the hidden layer. What we can infer from the Occam's Razor is that the number of hidden neurons should depend on the complexity of the function we want to estimate. Few hidden neurons are good for simple problems, more neurons are better for harder problems. We also know the the number of hidden neurons should be proportional to the input and output neurons. In fact, the expressive capability of a network is related with the number of parameters to be estimated, i.e. the number of weights:

$$N_w = N_i \cdot N_h \cdot N_o, \tag{2.5}$$

where N_i is the number of input neurons N_o is the number of output neurons. In addition, the number of training instances should be considered: the number of examples should be bigger than the number of parameter we want to estimate. As a rule of thumb, at least a dozen examples for each weight is often a sensible choice.

Acting on the activation functions may be useful or even necessary for some problems. While the logistic sigmoid works well for most of the problems, it can deal only with data normalized in the range [-1,1]. For example, linear activation function in the input and output layers allows to deal with every range of values. It may also be the best choice if we are sure that the function to be estimated can be well approximated as a linear combination of hyperplanes.

Monitoring the error on an unrelated validation set permits to adopt a robust stop criterion, optimizing the training while avoiding overfitting. A representative part of the training examples (e.g., a random 10%) must be in this case sacrificed to have a proper validation set. Validation tests can be performed regularly (for instance, at the end of each training iteration), and, after a first phase in which training error and validation error decrease together, the error on the validation error begins to rise, while the training error continues to decrease. Monitoring the validation error allows to detect the point of minimum error on the validation set. A simple criterion is to stop the training when the error begins to rise. A more robust choice, especially for hard problems in which the error could have many oscillations, is to store the weights when a minimum is encountered and let the training go until a safe point of no return is reached.

Encoding of the input data, although conceptually distinct from the design of the network, is probably the most crucial task for real-world problems. There are infinite ways the same information can be expressed in a vector of features, and they can be more or less suited for the proper training of the network. Input encoding should be defined by an expert of the problem, including all the information which is relevant for the input/output relationship. According to the Occam's Razor, less is better also in this case (the number of elements in the input vectors determines N_i), providing that no relevant information is lost. We know that the input data should be, explicitly or implicitly, correlated with what we want to predict. Unfortunately, also in this case, we don't have a general measure or rule which says us what is the best choice.

In conclusion, the design and training of a system based on multi-layer perceptrons involves the set up of some parameters and strategies which cannot be determined univocally. A common approach is to combine experiments and set-up, trying different parameters and architectures and choosing the best according to the experimental evidence.

2.5 Protein Structure Prediction

Predicting protein structure from sequence is one of the biggest open research issues in (computational) biology. In principle, the Anfinsen's dogma (13) makes us confident about the fact that the primary structure of a protein, i.e., the corresponding

sequence of amino acids, is all we need to infer the folding (at least for a globular protein). Molecular simulation of the folding process, which essentially consists of energy minimization in a water environment, seems to be a straightforward way to solve the problem. Unfortunately, according to the experiments performed by Levinthal (14), the folding process has on average 3^{300} degrees of freedom, thus generating a number of alternatives still intractable in computer simulations. This enormous difference between the actual speed of the folding process and the computational complexity for evaluating the corresponding model is also called Levinthal's paradox. Ab initio predictors that use some heuristics for reducing the search space have been developed, but the uncertainty about the degree of approximation of the actual structure limits their use only to very short chains or small perturbations around a known structure.

To overcome the absence of a deep model able to solve the folding problem, with modeling at the molecular level still intractable, searching in public databases, such as the PDB, is the first thing that a researcher usually does while looking for a protein structure. Unfortunately, the chances of finding the target protein listed in the PDB are not so high. In fact, notwithstanding the increase in experimental data on protein structures, the gap between known sequences (about 13 millions entries in UniProt in April 2010) and known tertiary structures (over 60,000 entries in PDB in May 2010) is exponentially increasing. As a consequence, the researcher can rely on different tools/techniques for protein structure prediction that have been developed to fill this gap (for a comprehensive review, see (44)). Given a target protein, the choice about which technique should be adopted to deal with the folding problem is mainly related to the observed evolutionary relationship with known structures. When no clear or reliable homology relationship can be detected, most methodologies, mostly pertaining to the field of fold recognition, make use of intermediate results, including secondary structure prediction.

2.5.1 Comparative Modelling

The difference between the number of protein sequences translated from sequences held in GenBank and the number of protein structures held by the PDB is vast. Only recently have high throughput methods started to be put in place to solve protein structure. Comparative modelling(45) offers a way to bridge the gap between the number of sequences and structures.

Comparative modelling generally relies on knowing the structure of a homologous protein and using that as a template to build the structure of a protein of known sequence but, unknown structure. The process can be divided into seven major steps: (i) identify homologous 'parent' structures to use in the modelling, (ii) align the target sequence with the parent or parents, (iii) identify structurally conserved regions (SCRs) and structurally variable regions (SVRs), (iv) copy the SCRs from the parent structure(s), (v) build the SVRs either by database search (e.g. SLoop(46; 47)) or *ab initio* methods (e.g. CONGEN(48)), (vi) build the sidechains(49; 50; 51; 52; 53; 54), (vii) optimize (e.g. energy minimization or molecular dynamics using software such as CHARMM(55) or NAMD(56)), evaluate (e.g. using PROSA II (57)) and refine the model. Methods such as COMPOSER(58; 59; 60) and SwissModel(61; 62) automate some of these steps. Another popular and effective method is MODELLER(63; 64) which combines stages (iii–vi) with optimization using restraints derived from the parents. There are many other methods including 3D-JIGSAW(65), FAMS(66), ESyPred3D(67) and RAPPER(68).

However, the limiting factor in all these methods is obtaining the correct alignment. This is the most important stage of comparative modelling(69; 70), but unfortunately, particularly at low sequence identity, it can be the most difficult to get right. The sequence alignment one wishes to achieve is the alignment that would be obtained by performing a structural alignment and reading off the resulting sequence alignment. While multiple alignment can help in obtaining the correct alignment, the structural alignment can often differ from the alignment obtained by performing global or local sequence alignment.

2.5.2 Fold Recognition

Fold-recognition (often, but not always properly, referred also as *threading*) methods are applied when no homologous protein can be reliably identified; the typical case is that the best match in the PDB is below 25% sequence identity. The main assumption of fold-recognition is that the number of protein folds (i.e. spatial arrangement of secondary structure elements) is relatively small (71; 72). Typically, a fold-recognition program is based on four main components:

1. Representation of the template structures. This is usually achieved through a search in the PDB (Section 2.1.3) or some classification of protein structures, such CATH or SCOP (Section 2.1.4).

2. Evaluation of the compatibility between the target sequence and a template fold. This is achieved by defining and applying a objecting function.

3. Alignment between the target sequence and the template structures with the chosen fold. Dynamic programming or other alignment techniques can be used for this task.

4. Select the best alignment (ranking) according to the objective function and build the model by atom substitution.

Many fold recognition methods have been proposed in the literature. The main distinction is between those that detect sequence similarity and those that detect structure similarity. A simple sequence-only fold recognition operation is to use BLAST or PSI-BLAST to search the PDB for proteins that exhibit significant sequence similarity to the target protein. Modern sequence-based fold-recognition methods utilize the evolutionary information available both for the target and the template (73; 74).

Structure-based fold-recognition, often (in this case properly) referred to as threading, utilizes the experimentally determined structural information from the template. The target sequence can be enhanced by including predicted structural features of the target, including secondary structure elements, solvent accessibility, polarity of the side chain environment and local backbone conformation (75; 76; 77; 78; 79; 80; 81; 82; 83; 84; 85).

Rosetta (86) is an effective hybrid de-novo/fold-recognition method based on the distribution of conformations of fragments of three-and nine-residue segments.

2.5.3 Ab initio Prediction

Ab initio –or *de novo*– predictors tries to obtain folding conformation without the use of other structure templates. In order to predict tertiary structure, protein folding energetics or statistical tendencies can be used to build target functions to be minimized. Owing to the vast conformational space to be explored, ab initio techniques require high computational resources thus limiting their use to small sequences. Generally, rather

than trying to directly model the tertiary structure, ab initio prediction is reduced to the prediction of (i) one-dimensional features, such as solvent accessibility (87; 88), disordered regions (89), secondary structure (see Chapter 3) (ii) two dimensional features, such as residue contact maps (90; 91). Recent promising advances in the prediction of tertiary structure have been obtained with the use of coarse-grained protein models growing in their native environment and secondary structure prediction (92).

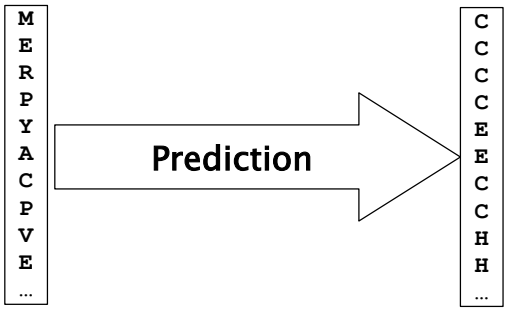
Chapter 3

Secondary Structure Prediction

The knowledge of the secondary structure has proven to be a useful starting point for investigating further the problem of protein tertiary structure (see Section 2.5), function (93; 94), and to improve sequence alignments (95). In this chapter, the problem of secondary structure prediction (SSP) is presented and analyzed. The characteristics of the SSP problem are first described from a computational point of view (Section 3.1), and the literature in the field is presented from a historical perspective (Section 3.2) .Then, the standard evaluation measures and assessment criteria are detailed (Section 3.3). Finally, the SSP problem is analyzed: five different sources of information are identified and discussed (Section 3.4), and some well known techniques are reinterpreted with this new insight (Section 3.5).

3.1 Characteristics of the SSP problem

From a computational point of view, predicting a protein secondary structure consists of assigning a secondary structure label to each residue of a given protein chain:



We can see a primary structure as a signal to be decrypted to know the structure signal. In the absence of a model able to unveil the relationship between this two signals, methods relying on statistical properties, typically belonging to the fields of machine learning and pattern recognition, are generally applied to SSP. As a machine learning problem, the most notable characteristics of SSP are as follows:

- *Great input space:* In a typical setting, i.e., sliding window of length 15 with a PSSM encoding, the input dimension is $15 \times 21 = 315$[1].

- *Large training set:* PDB available structures are about 60,000. Upon the removal of homologous sequences, a typical dataset contains about 5000 proteins. Assuming an average length of 250 residues per protein, the overall number of data samples is more than 1 million, and many more should be needed to properly represent the prediction space. The dimension of the training set is clearly a limitation for some computational methods: predictors that require all samples to be stored in memory at the same time cannot be trained with all the available data.

- *Low input/output correlation:* Although increased by injecting information about multiple alignments (in the representation of inputs), the correlation is still very difficult to identify. See Section 3.4.

- *Discrepancy between training set and test set:* One of the main postulates of machine learning is that training inputs must be highly representative of the whole input space; the higher the representativeness, the better the expected performance of a classifier or predictor is. However this assumption must be neglected to some extent because for SSP, in ab initio contexts we are only proteins whose structure is not related from any other known structure are considered.

- *Labeling noise:* Owing to protein mobility, measurement and/or automatic labeling errors introduce some aleatory behavior superimposed on the "correct" data used for learning (see Section 1.4.3.1).

[1]The additional value (21 instead of 20) comes from the fact that an extra input is typically used to encode positions that lay outside the sequence. See Section 3.4.2

3.2 Secondary Structure Predictors: History and State-of-the-art

Secondary Structure Prediction is one of the historical problems in bioinformatics; the first attempts date back to the half sixties (96), and a plethora of methods have been proposed to tackle the problem from then on. After a peak of interest and innovations during the nineties, a reasonable (but still not really satisfying for many applications) accuracy has been reached by the state-of-the-art. In early times, although the interest in the field has lowered (mainly for the lack of important performance improvements), research in the field is still active. The history of secondary structure predictors is generally divided into three phases, or generations.

3.2.1 First Generation Predictors

Early prediction methods rely on the propensity of amino acids to belong to a given secondary structure. Relevant methods proposed were the Chou-Fashman(97) and the GOR (98) .

In the Chou-Fashman method, the relative frequencies of each amino acid in each secondary structure of known protein structures are used to extract the propensity of the appearance of each amino acid in each secondary structure type. Propensities are then used to predict the probability that a given sequence of amino acids would form a helix, a beta strand, or a turn in a protein. *alpha*-helices and *beta*-strands are predicted by setting a cut off for the total propensity for a slice of four residues. A similar, but more sophisticated, criterion is adopted for turns. The original Chou-Fashman parameters were derived from a very small sample of protein structures due to the small number of structures available at the time of the work. It reached about 50% accuracy.

The GOR method is another propensity-based method which introduces the conditional probability of immediate neighbor residues in the computation. It uses a window of 17 residues and it is based on Bayesian theory. The method has been refined several times, finally reaching about 64% accuracy (99).

3.2.2 Second Generation Predictors

A second generation of predictors exhibit better performance by exploiting protein databases as well as advanced statistical and machine learning techniques. Several methods exist in this category that can be classified according to: (i) the underlying approach, including statistical information (100), graph-theory (101), multivariate statistics (102), K-Nearest Neighbors (103), and ANNs (104; 105; 106); (ii) the kind of information actually taken into account, including physic-chemical properties (107) and sequence patterns (108); The accuracy of second generation predictors is approximately in the range 63-68%.

3.2.3 Third Generation Predictors

The turning point to third generation methods was the idea of exploiting evolutionary information to feed learning systems. Multiple alignments encoded as frequency profiles or hidden markov models proved successful with ANNs: (109; 110; 111; 112; 113; 114). Other techniques have also been experimented, including bidirectional recurrent ANNs (115; 116), linear discriminants (117; 118), nearest-neighbours (119; 120), HMMs (35; 121), Support Vector Machines (SVMs) (122; 123). Hybrid approaches include meta-predictors (124), populations of hybrid experts (125), ANNs combined with GOR and linear discriminants (126), a knowledge base (127), dynamic Bayesian networks (128).

Third generation predictors have an accuracy ranging 72% to 80%, depending on the method, the training and the test datasets.

3.2.4 Secondary Structure Prediction Servers

Secondary structure predictors, as well as most bioinformatics tools, are often made available as servers. Table 3.1 shows some of the most famous secondary structure prediction servers.

3.3 Performance Assessment

Performance assessment of secondary structure predictors has been a main and, for a long time, controversial issue. In this section, the different aspects related to this issue are examined. In particular, the most relevant standard measures, testing issues, and expected improvements in the field, are discussed in the next paragraphs.

Program	Reference	URL
Three-state prediction		
APSSP2	G.P. Raghava (unpubl.)	www.imtech.res.in
HMMSTR	Bystroff *et al.* (121)	www.bioinfo.rpi.edu
JPRED3	Cuff *et al.* (129)	www.compbio.dundee.ac.uk
NNSSP	Salamov and Solovyev (119)	bioweb.pasteur.fr
PHD	Rost *et al.* (109)	cubic.bioc.columbia.edu
PORTER	Pollastri and McLysaght (116)	distill.ucd.ie
PRED2ARY	Chandonia and Karplus (110)	www.cmpharm.ucsf.edu
PREDATOR	Frishman and Argos (120)	ftp://ftp.ebi.ac.uk
PROF	Ouali and King (126)	www.aber.ac.uk
PSIPRED	Jones (113)	bioinf.cs.ucl.ac.uk
SSPRO	Pollastri *et al.* (115)	www.igb.uci.edu

Table 3.1: Relevant secondary structure prediction servers.

3.3.1 Kinds of Errors and Standard Evaluation Measures

Prediction errors can be divided in two main categories: (i) *local errors*, which occur when a residue is wrongly predicted, and (ii) *structural errors*, which occur when the structure is globally altered. The most unwanted errors are the latter, with secondary structure segments as the basic components of the three-dimensional structure of a protein. In particular, errors that alter the function of a protein should be avoided whenever possible. Unfortunately, they cannot be easily identified; thus, local measures or generic segment superposition measures are usually adopted. Q_3 and Matthews Correlation Coefficients (C_h, C_e, C_c) are commonly used measures of local errors, whereas the Segment Overlap Score (SOV) is the most well known measure for structural errors. These measures have been adopted in the context of CASP (130) and EVA (131). They will be reported as follows for the sake of completeness (e, h, and c stand for alpha-helices, beta-sheets, and coils, respectively).

- Q_3. It is a measure of accuracy, is largely used for its simplicity, and accounts for the percent amino acids correctly predicted. It is defined as follows:

$$Q_3 = \frac{\sum_{i=h,e,c} Q_{3i}}{3} \tag{3.1}$$

$$Q_{3i} = 100\frac{t_{p_i}}{N_i}, \qquad i \in \{h, e, c\} \tag{3.2}$$

where N_i is the number of residues observed for structure category i, and t_{p_i} (i.e., true positives) is the corresponding number of correctly predicted residues.

- *SOV*. The Segment OVerlap score (132) accounts for the predictive ability of a system by considering the overlapping between predicted and actual structural segments. It is able to consider structural errors where predictions deviate from experimental segment length distribution. The definition of the SOV measure for class i is as follows:

$$SOV_i = \frac{1}{N_i} \sum_{s_i} \frac{minOV(s_1, s_2) + \delta(s_1, s_2)}{maxOV(s_1, s_2)}. \tag{3.3}$$

Here, s_1 and s_2 are the observed and predicted secondary structure segments in the i state; S_i is the number of all segment pairs $(s_1,\ s_2)$, where s_1 and s_2 have at least one residue in i state in common, $minOV(s_1,\ s_2)$ is the length of the actual overlap of s_1 and s_2 and $maxOV(s_1,\ s_2)$ is the length of the total extent for which either of the segments s_1 or s_2 has a residue in i state. N_i is the total number of elements observed in the i conformation. The definition of $\delta(s1,\ s2)$ is as follows:

$$\delta(s_1,\ s_2) = \min \begin{cases} maxOV(s_1,\ s_2) - minOV(s_1,\ s_2) \\ minOV(s_1,\ s_2) \\ int\ (0.5 \times len(s_1)) \\ int\ (0.5 \times len(s_2)). \end{cases} \tag{3.4}$$

The total SOV measure can then be obtained by taking the average for the three classes h, e, c weighted with the number of residues belonging to each class.

- C_h, C_e, C_c. Defined in (133), the Matthews Correlation Coefficient (MCC) relies on the concept of *confusion matrix* (Table 3.2). Confusion matrix is a general method to visualize classification errors generally adopted in supervised learning. A table represents the results of a binary classification: columns represent the predicted class, while rows represent the actual class. One benefit of confusion matrices is that they make it easy to see when the system is confusing two classes (i.e., commonly mislabeling one as another).

  The MCC is a synthesis of the information contained in the confusion matrix, able to take into account the ability of the evaluated system in not confusing

class	*predicted positive*	*predicted negative*
positive	true positives (t_p)	false negatives (f_n)
negative	false positives (f_p)	true negatives (t_n)

Table 3.2: Confusion Matrix. Elements on the main diagonal represent correct classifications, elements out of the diagonal represent errors.

a class with the other. Evaluated on a specific secondary structure, MCCs are defined as follows:

$$C_i = \frac{t_{p_i} t_{n_i} - f_{p_i} f_{n_i}}{\sqrt{(t_{p_i} + f_{p_i})(t_{p_i} + f_{n_i})(t_{n_i} + f_{p_i})(t_{n_i} + f_{n_i})}}, \quad i \in \{h, e, c\} \tag{3.5}$$

where t_{p_i} represents the number of true positives, t_{n_i} the true negatives, f_{p_i} the false positives, and f_{n_i} the false negatives. Note that MCC is not defined in the event that no occurrences of class i are found.

3.3.2 SSP Testing: Which Proteins Should be Used to Test SSP Sytems?

Another fundamental issue is about which protein sets should be used to test the predictors. The machine learning theory suggests that test samples must be different from the training samples, which in turn are expected to represent "well" the concept to be learned. This definition only partially fits the requirements that apply to SSP. The presence of homologous proteins may alter the perceived performance, being predictors used when no homologous templates are available. If a homologue is available for the target protein, its structure can be usually assigned without the need for a secondary structure predictor. Hence, a secondary structure predictor should be tested with proteins that are not homologue with any of those used for training. In principle, this definition requires the analysis of protein families, throughout resources for structural classification (see Section 2.1.4). Usually a safe threshold of 25% sequence identity against proteins used in the training set is applied while building the test set. This concept has become clear during CASP, the annual conference whose purpose is to assess the advances in protein prediction. After CASP4, automated testing has been considered to be more appropriate for secondary structure predictors, giving birth to the EVA server, which automatically asks well-known prediction servers to predict target proteins before their actual structure is made publicly available.

3.3.3 Presentation of Results: Macro vs. Micro Averaging

While reporting experimental results, an average for Q_3, SOV, and $MCCs$ on a given test set is usually reported. Two kinds of averages can be computed: (i) *by chain*, in which measures are computed for every single chain and then averaged on the test set, and (ii) *by residue*, in which measures are computed for the whole test set, as if they were a big single chain (alternatively, it can be considered an average-by-chain weighted by chain length).

The average-by-chain usually leads to lower scores, when short proteins are more difficult to predict. Further problems may arise while evaluating SOV and $MCCs$.

Average-by-residue scoring does not make sense for the SOV measure, although it may be equivalently computed as a by-chain average, weighted with the length of the sequences.

As for MCC, the average-by-chain has to be interpreted with care: when the MCC is not defined, the maximum value (1.0) is usually taken, leading to an average whose absolute value does not reflect the real performance for the different structure categories. In particular, when average-by-chain is applied, the C_e may be larger than the C_h value, giving the impression that the beta-sheets are easier to predict than the alpha-helices. However, this is only a distortion due to the fact that the C_e value is undefined with an occurrence higher than C_h.

3.3.4 Asymptotic Limit and Expected Improvement

Rost (134) asserted that the limit on the maximum Q_3 obtainable in SSP is about 88%. This limit is mainly due to the intrinsic uncertainty about structure labeling, which in turn is related to the dynamics of the target protein structure and to the thresholds used by standard programs (e.g., DSSP (10)) to perform the labeling task. While significant improvements have been obtained since the first predictors, no relevant steps forward have been made in the last 10 years to reach the cited limit. Obtain improvements of even a fraction of a percent has become challenging. Current state-of-the-art predictors claim a performance of about 77%–80% Q_3 and 73%–78% SOV, depending on the dataset adopted for testing. As for the structures to be predicted, alpha helices ($C_h = 0.7 \div 0.77$) generally appear to be far easier to predict than beta strands ($C_e = 0.64 \div 0.68$).

3.4 Informative Sources in the SSP Problem

The success of a modern predictor depends on its ability to exploit the correlation between the sequence and the structure. Owing to the great variability in the possible configurations of sequences, and also to problems related to the representation of variable-length sequences, only relying on intrinsic sequence-to-structure correlation is a poor strategy to obtain good predictions. Fortunately, other kinds of correlation hold in protein sequence and structure, and finding the way to exploit the related information determines the success of a modern predictor. In the following analysis, five different kinds of correlations are identified, showing how they are exploited in actual prediction systems.

3.4.1 Sequence-to-Structure

Sequence-to-structure correlation is related to the probability of observing a structure S in the contest of a given sequence P.

The correlation between a sequence –considered in its entirety– and its corresponding structure is clearly high: Anfinsen's dogma (13) states that the tertiary structure of a globular protein is completely determined by its primary structure. One limit to its use in the prediction task is related to the high specificity of protein sequences: sequences can vary both in length and composition, leading to a virtually infinite[1] set of alternatives, each one –at least in principle– with its own structure. Looking at *real* proteins the number is critically lower, because only a small part of the theoretically possible sequences are actually likely to fold, and not all possible protein folds have been chosen by evolution. In any case, the number of possible sequences is still high; about 12 million sequences are known, and the number is increasing exponentially. The known structures, i.e. patterns usable to extract statistics or train learning algorithms, are about three orders of magnitude lower, and mostly concentrated on small clusters of proteins of high biological or medical interest.

The variable length of the protein chains also makes the sequence-to-structure information difficult to exploit, since most known algorithms deal with data of fixed length (most pattern recognition and machine learning methods are based, directly or

[1] There are 20^l theoretically possible sequences of length l, where l can range from some dozens to a few thousands residues.

indirectly, on a distance measure in the input space, represented with vectors). Fortunately, sequence-to-structure correlation is also conserved in fragments of sequence, permitting these methodologies to be applied to local fragments.

A weak correlation can be observed between the type of a residue and the secondary structure to which it belongs, and the first propensity-based prediction methods relied on this assumption. A stronger correlation holds between a fragment of chain and the corresponding structure; for this reason, most predictors consider the surroundings of the residue to be predicted. Slices of primary structure (extracted with a sliding window) can be then considered to represent the primary information in a suitable format. Enlarging slices may be assumed to facilitate the task of learning the actual I/O relation, but larger slices do not necessarily entail better predictions. Whereas a larger slice may actually highlight the correlation between input and output, the generalization capability of a predictor may be negatively affected (for further discussion about slice size, see (135)). Usually, sliding windows of 5-21 residues are considered depending on the underlying algorithm.

Many different machine learning or pattern recognition techniques have been applied to estimate the relationship between primary and secondary structure. Any classification algorithm can be used to predict a SS elements from encoded slices; among all, multi-layer perceptrons have been shown to be particularly effective. Alternative techniques have been proposed: for example, the recurrent neural network architecture proposed by Baldi *et al.* is able to take into account far sequence-structure relationships (see Section 3.5.3).

3.4.2 Inter-Sequence

Given two sequences P_1 and P_2, the inter-sequence correlation is related to their similarity, measured for example counting the percentage of identical residues, or more generally applying a scoring function on their pairwise alignment.

Basically, inter-sequence relationships are the result of evolution, which have selected the most useful sequences according to their folding. Taking into account the evolutionary relationships (the lower the evolutionary distance, the higher the correlation), families of proteins with similar sequences can be identified.

In order to be exploited by a prediction algorithm, the inter-sequence correlation can be included in the representation of the sequence: encoding based on multiple alignment

(e.g. substitution frequency or HMM profiles) is a fundamental technique aimed at highlighting this kind of correlation, exploiting the available sequence databases. From the point of view of the predictor we can see the representation of protein families as a way of clarifying the distance relationships between proteins. In other words, inter-sequence relationships give us a new representation space in which proteins belonging to the same family become nearer, while unrelated proteins become more distant to one another.

Note that not all encoding alternatives are equivalent, owing to the loss of the original information contained in the multiple alignment during the encoding process. For instance, PSI-BLAST PSSM (34) is usually preferable to the raw frequency count, and other encoding methods can also be devised. A new way to encode multiple alignment information is presented in Chapter 5. The introduction and improvement of encoding techniques based on multiple-alignment is acknowledged to be the most important innovation in modern predictors (about +9% Q_3 improvement from the first systems based on ANNs to PSIPRED). Nowadays, almost every fold recognition method and secondary structure predictor makes use of encoding based on multiple alignment.

3.4.3 Inter-Structure

Similarly to inter-sequence correlation, inter-structure correlation is prevalently observed in presence of homology. Correlation among structures is more conserved than the one occurring for sequences, so further explaining the effectiveness of multiple alignments in describing protein families. In particular, closely related homologous proteins have very similar structure. Figure 3.1 shows the structural similarity averagely found in relationship with evolutionary relationship observed between sequences.

Comparative modeling techniques effectively exploit inter-structure information in presence of homology. Without clear homology relationships, inter-structure correlations still exist: it has been argued that very few actual folding templates have been observed compared with the number of unrelated protein families (71) and it is confirmed by structure classification resources such as CATH and SCOP. Fold recognition techniques make use of this kind of inter-structure correlation to predict the folding of new proteins.

Figure 3.1: Evolutionary relationship vs. structure similarity. The values have been obtained searching for 100 unrelated proteins in the PDB for similar sequences, and then aligning their secondary structures according to the alignment provided by BLAST. The BLAST e-value has been taken as measure of evolutionary relationship, structural identity is normalized in [0,1]. It can be noticed that sequence identity steeply decreases for e-values below 10^{-5}.

Being secondary structure prediction acknowledged to be an ab initio technique, 'proper' predictors should not make use templates to build their predictions. For this reason, the absence of a clear homology relationship with any known structure is normally considered a pre-condition for a fair evaluation of SS predictors. Furthermore, when an evolutionary related template can be found, it is probably more effective to assign the structure directly from the template rather than using that information to train a learning algorithm. Pollastri *et al.* (136) showed that using information of remote homologues, when available, can improve the accuracy in secondary structure prediction.

Correlation between structures can be also seen without looking for a template. Various preferences can be seen in the distribution of SS elements and structural features. This preferences can be exploited in prediction; for example, local preferences are indirectly taken into account by structure-to-structure (S2S) modules used by many predictors, and they have been used to build statistical models, such as HMMs and dy-

namic Bayesian networks, used in predictors.

A statistical analysis of inter-structure correlations observed in known proteins, and a proposal to exploit information found in structure distribution is presented in Chapter 7.

3.4.4 Intra-Sequence

Intra-sequence correlation, or 'auto-correlation' of the primary structure, is related to the possibility of guessing a protein's residue, given a subsequence of that protein.

In principle, a high intra-sequence correlation would greatly restrict the possible protein sequence configurations and, as suggested by information theory, would raise the possibility of having more compact representations of protein sequences. Unfortunately, although a compact representation is usually desirable in machine learning and pattern recognition algorithms (as the Occam's Razor principle suggests), the observable intra-sequence correlation is very low (137), and does not appear to have any direct application in any actual prediction technique.

3.4.5 Intra-Structure

Just as for primary sequence data, an auto-correlation of secondary structure labels can also be considered. A strong correlation holds within secondary structure elements and between beta-strands; it depends on the mutual interactions holding between the amino-acids along the protein, including the hydrogen bonds involved in the formation of secondary structure. Also when the interactions are local, this principle of locality is not necessarily observable at the sequence level.

Residues that belong to the same helix or to the same strand are –by definition– very close to each other also at the sequence level. Conversely, residues that belong to the same sheet but occur in different strands are typically not close at the sequence level. Taking advantage of this kind of correlation is not easy due to the local scope of prediction algorithms.

Refinement processing is a successful way to exploit the correlation within structures. Using a structure-to-structure (S2S) prediction module is a relevant alternative proposed in different actual systems able to deal with intra-structure correlations. Ac-

cording to our experiments[1], S2S refinement generally improves the accuracy of a SS predictor by about 0.5-1.5%.

Statistical models such as dynamic Bayesian networks and HMMs incorporate by construction some sequential intra-structure correlation.

We believe that actual systems are using only a small part of the information contained in intra-structure interactions. Alternative ways to use this kind of information dealing with output representation and prediction architectures are presented in Chapter 6.

3.5 Notable Predictors

3.5.1 PHD

PHD is an archetype for a successful family of systems based on ensembles. Each component of the ensemble is a pipeline of two transformations (both implemented by an ANN): primary-to-secondary prediction (P2S) and secondary-to-secondary prediction (S2S). The overall transformation (i.e., P2S-S2S) is depicted in Figure 3.2. In P2S, the

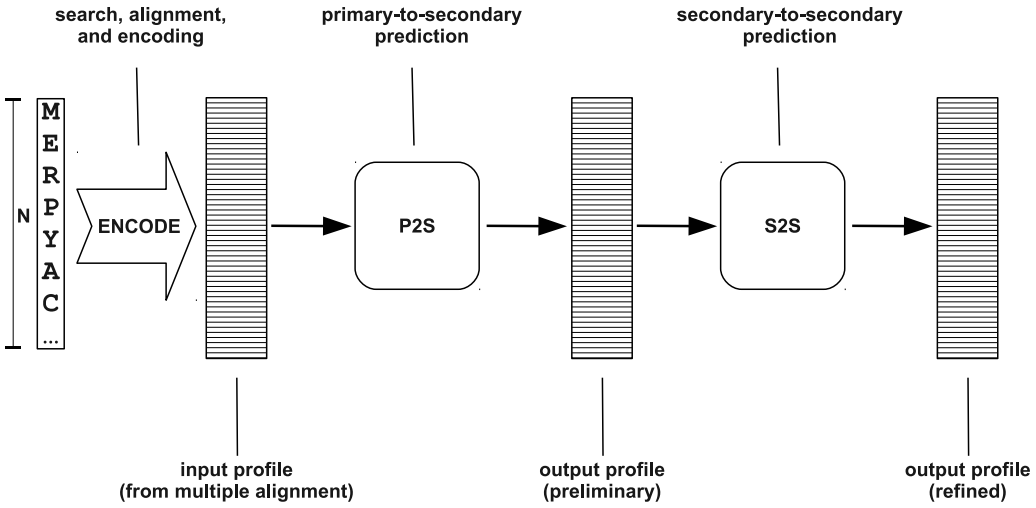


Figure 3.2: PHD: a schematic view of a P2S-S2S transformation

ANN is trained with slices extracted throughout a sliding window of fixed length from an input profile, which in turn is obtained by encoding the multiple alignment that

[1]Evidence can be seen comparing Figures 5.3 and 5.4.

represents the target sequence. PHD adopted frequency profiles obtained by the HSSP databank (138), and turned to alignments obtained with BLAST in later versions. It should be noted here that the prediction problem is actually turned into a classification problem due to the splitting of the target protein in fixed-length slices obtained using the sliding window. Thus, the central amino acid of a slice can be labeled in isolation, yielding a preliminary prediction obtained by annotating each amino acid of the target sequence with the results (position by position) of the classification.

In S2S, the ANN is trained with fixed-length slices generated by the P2S module (again, through the use of a sliding window). The S2S transformation is also called refinement. In so doing, the problem is moved back from classification to prediction; information about the correlation that holds amino acids belonging to the same secondary structure is taken into account, to some extent, while performing S2S refinement.

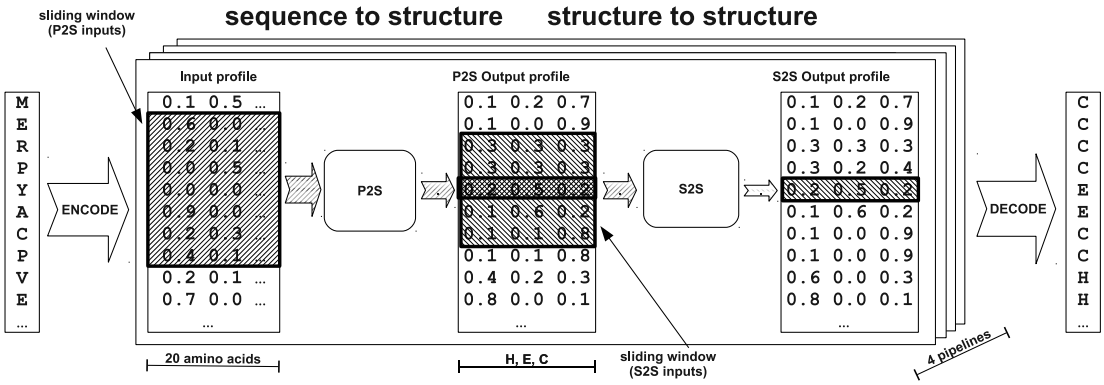


Figure 3.3: PHD: Overall architecture

As multiple P2S-S2S transformations are involved in a prediction (to enforce diversity), further level of combination is performed on the outputs of the P2S-S2S available pipelines. The actual prediction is issued by decoding the resulting (averaged) output profile with a maximum-a-posteriori probability (MAP) criterion. PHD adopts a jury decision after performing an average-by-residue on output profiles. A schematic view of the overall architecture (with some details about the use of sliding windows) is shown in Figure 3.3.

In sum, the encoding process, the P2S transformation, and the S2S transformation can be considered successful attempts to deal with sequence-to-sequence, sequence-to-structure, and structure-to-structure correlation, respectively. Furthermore, at least in

our view, the PHD architecture performs a two-level unfolding of a fixed-point strategy for secondary structure prediction.

3.5.2 PSIpred

PSIpred (113) is widely acknowledged as one of the most effective secondary structure predictors, inspired to PHD. The main innovation of PSIpred was to use the standard PSSM matrices used by PSIBLAST to encode proteins. Its success can be then explained with the new effective way to use inter-structure correlation information.

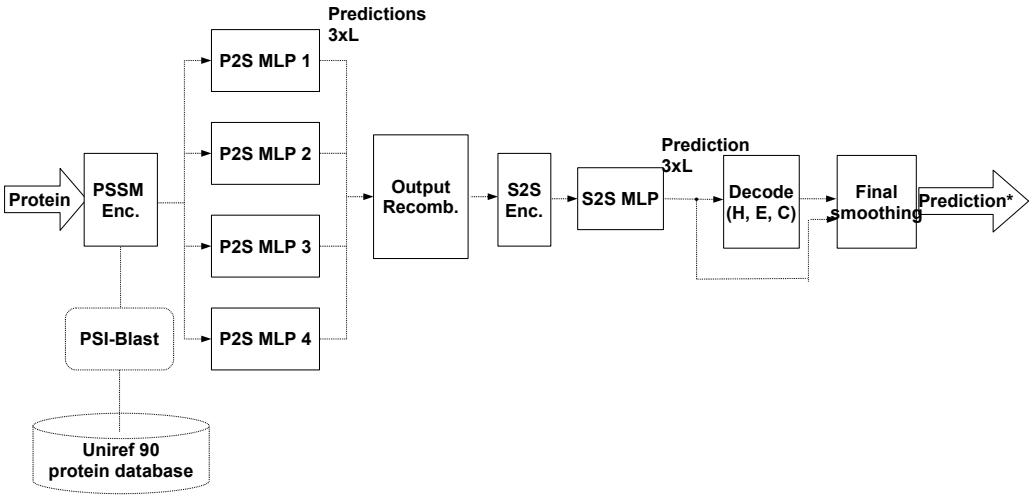


Figure 3.4: PSIpred architecture

Figure 3.4 sketches the architecture of PSIpred[1]. A sliding window of 15 positions is used to feed the networks with the information found in the PSSM profile. In particular, four MLPs are independently trained to predict a 3-bit vector, used to assert the label of the central residue of the sliding window (primary-to-secondary structure prediction). Their predictions are then combined by means of a suitable MLP, entrusted to issue the final prediction (secondary-to-secondary structure prediction). The input of this second-layer expert is constructed upon the average of the predictions issued by the first-layer predictors. A training input for the S2S layer includes additional global information: the average of the predictions given for each structure type, the protein length and the relative position of the sliding window within the chain.

[1]Compliant with the version 2.6 of PSIPRED, as downloaded from http://bioinf.cs.ucl.ac.uk/downloads/psipred/

In the post-processing, some biologically-inspired rules are used to find and revise parts of the prediction that are in fact not likely or impossible to occur. For example, single helices are removed, and single structures in general are disfavored. Following the analysis proposed in Section 3.4, this kind of post-processing can be viewed as a way to exploit the auto-correlation within structures.

3.5.3 SSPRO

SSPRO (115) is a predictor whose distinctive characteristic is the use of bidirectional recurrent neural networks (BRNN). BRNNs are adopted in order to overcome the limits of MLPs in considering only a window of fixed size to perform the prediction. This is achieved with three components (Figure 3.5): the central component is related the

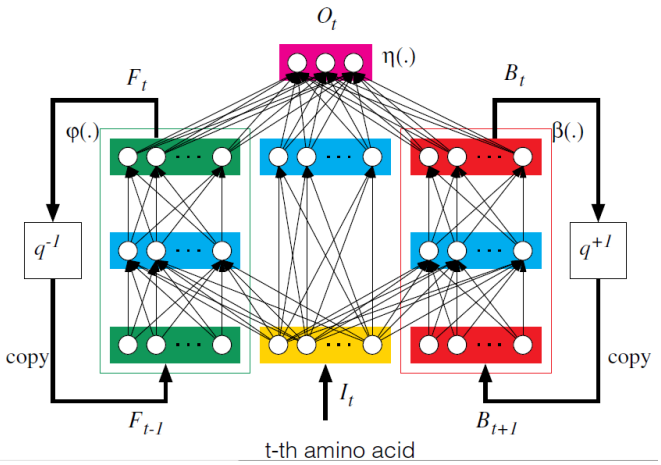


Figure 3.5: The BRNN architecture adopted by SSPRO.

residue to be predicted, the other two components are related to the forward and the backward rest of the protein chain, represented by a BLAST or PSI-BLAST profile. Eleven BRNN, trained separately with different parameters, are then combined to obtain the final prediction.

The approach adopted by SSPRO can be viewed as an attempt to enhance the sequence-to-structure correlation: the presence of the forward and backward nets permits to take into account a wider context than permitted by a simple sliding window. On the other hand, the improvements obtained with the use of BRNN are limited by a more difficult training. In fact, according to the EVA automatic evaluation, although

cutting-edge, the performance of SSPRO did not really overtaken other simpler neural network methods like PSIPRED. Porter (116) is an evolution of SSPRO with enhanced input encoding, output filtering, bigger ensembles and, in later version, structure assignment from homologues (where available)(136). Porter is the system reported to have the best performance in the EVA experiments[1].

[1]It is worth noting that the assessment of the performance of Porter is in part controversial due to the use of the structural assignment: the exploitation of inter-structure information is in fact excluded by the competitors (see Section 3.4).

Chapter 4

GAME: a Generic Multiple Expert Architecture for Real-World Prediction Problems

Machine learning is a branch of artificial intelligence concerned with the design and development of algorithms trained on empirical data. Machine learning techniques can be adapted to a wide variety of prediction problems related to the fields of data mining and pattern recognition. A particularly challenging category of problems are "real world" problems, i.e. problems arising from real measures in a real environment.

Real-world problems (RWPs from now on) may be related to human activities, like text categorization or natural language processing, and/or interaction with the environment, like obstacle detection for automatic driving or robot movement. Many RWPs arise from biometrics and bioinformatics.

RWPs are opposed to "toy" problems (TPs), which may be games or abstractions of real world problems. Typical TPs classification problems define a hyperspace of features and two or more subspaces within it that have to be recognized one from each other. The same definition is also appropriate for RWPs classification problems, after a first phase of *feature extraction* and *labeling*, with the difference that the exact boundaries of these subspaces are unknown.

More than the intrinsic complexity what distinguishes RWPs from TPs is the knowledge of the environment, which can be known with infinite precision in TPs. In RWPs measurement and labeling are uncertain, the amount of available examples is limited

and the distribution of the examples is not controlled. Then, the main issues are different: the prediction/classification algorithm is the central issue in TPs, whereas feature extraction and labeling are often the most critical issues for RWPs. The limited training data available –and its sampling– is another issue in the design of ML systems for RWPs.

Given a set of labeled data, many machine learning algorithms are able to give an estimation of the input-output relationship, and some algorithms –or combinations of them– may be more suited than others depending on the characteristics of the space of features. For example, a binary classification can be dealt with Bayesian classifiers, multi-layer perceptrons (Section 2.4), support vector machines (139), nearest-neighbors, decision trees and many other techniques[1]. Meta learning techniques, like bagging(140) and boosting(141), are general techniques for building ensemble classifiers from base classifiers. ECOC decomposition (142) permits to build ensembles for multi-class classification from binary classifiers.

What algorithm is the best is not known a priori, and most algorithms have to be properly configured to work at their best. Human expertise must be often integrated with iterative optimization, training and evaluating the system with different parameters ("test and select" approach (143; 144)). Specific software tools help running different algorithms and configuration and assessing them with a graphical or scripting interface.

Widely acknowledged tools of this kind are WEKA (145) and RAPIDMINER (146). The MATLAB® (© the MathWorks) environment, with the help of specific toolboxes, is also widely used within the data mining and pattern recognition communities. All these tools provide wide support for the algorithmic part of the work, and almost every known algorithm is made available. They can load data in various formats, provide data pre-processing facilities and allow to build ensembles graphically, programmatically, and through meta-learning algorithms.

A limit of these tools is that they operate at a level in which both the feature extraction and labeling tasks have been accomplished. This can be a big limitation in all these cases in which the input features are obtained after a complex extraction from structured data, such as texts, signals, images, or sequences. All kinds of data that we commonly see in real-world problems. In this cases, other pieces of software must be

[1]See Duda *et al.* (38) for a comprehensive guide about pattern classification.

written separately do deal with feature extraction, using specific techniques or general approaches like extraction with sliding windows or similar. Although may not necessarily be bad having a separate pre-processing that builds the datasets needed, it may be a big limitation when the pre-processing have to be designed and tested together with the rest of the system. The feature extraction task is frequently more difficult and with more open question than the prediction itself. With a uniform approach feature and labeling may be parametrized in order to take advantage of comparative experiments together with the prediction task. The presence of a separate pre-processing can be also annoying when a released of the prediction system is required, for example as a stand-alone application, an independent API, a web service or a server.

Another limitation is related to the explosion of memory requirements caused by the duplication of data due to the extraction with sliding windows from structured data. In fact, the cited tools deal with vectors of features which are loaded statically, and running out of memory is quite common. For example, a single 250×250 image encoded with 8 bits goes from the original 62.5 KB up to 4 MB occupied using 8×8 sliding windows, and thousands of such images may be available for the training.

Secondary structure prediction, previously described in Section 4.3, is one of those problems which unveil the limitations of common machine learning tools and can take advantage of a new approach tailored for real-world problems. In particular, studies about input encoding, output encoding and prediction architecture can take advantage of a modular architecture which allows to experiment all of the related issues in the same framework. These needs, together with the need of release prediction servers and stand-alone programs, leaded to the development of a new software architecture and framework, GAME.

GAME is a software architecture and a framework written in Java that supports rapid prototyping and testing of classifiers and predictors for real-world problems, from a multiple-expert perspective. It provides a graphical environment that allows to configure and connect automated experts (i.e., classifiers or predictors), so that non-experienced users can easily set up and run actual systems. The version of GAME described in this document is 2.0, an evolution of the first release described in (8) and (9). The main features of GAME are:

- *Plug-in Architecture.* Critical modules, described by simple interfaces, can be loaded and interchanged at runtime, with the help of a graphical user interface XML descriptors.

- *Support for real-world problem modeling.* GAME allows to model the overall transformation required to solve a given problem from raw data to a readable prediction. Data are loaded, coded with one of the available (or created ad-hoc) modules, processed, and then decoded in a usable format.

- *Support for comparative experiments.* A graphical interface allows to incrementally set up of batteries of experiments whose results are automatically logged for separate analysis.

- *Support for expert combination.* Experts are defined as autonomous entities. They can be combined in multiple ways, making it easier to devise and implement actual systems characterized by complex software architectures. GAME allows to devise various software architectures, recursively combining together and/or refining experts defined separately.

- *Just in time dataset construction.* The possibility of defining just-in-time dataset iteration prevents a system from running out of memory also with large amounts of structured data. A caching mechanism prevents the useless performance worsening. Just-in-time datasets permit also to implement training strategies with synthetic or resampled data which changes at every iteration.

- *Portability.* Portability is guaranteed with the use of the Java language and strict design requirements. A system built with GAME does not require any kind of installation, and can be directly run from a simple unpack and run bundle.

- *Support for final release.* Data is handled in its natural format. As consequence, prediction systems built with GAME are ready to be deployed and used in a real environment with very few adaptations. Furthermore, any component of a system built with GAME can be serialized and loaded with both XML and the Java-integrated serialization API. Thanks to these features and to the full portability the release of a prototype is straightforward for all the common operative systems.

4.1 The Generic Architecture

4.1.1 GAME Approach to Prediction

The core prediction machinery of GAME is based on the decomposition of a problem in five independent actors (Figure 4.1), which define a problem by its input and output

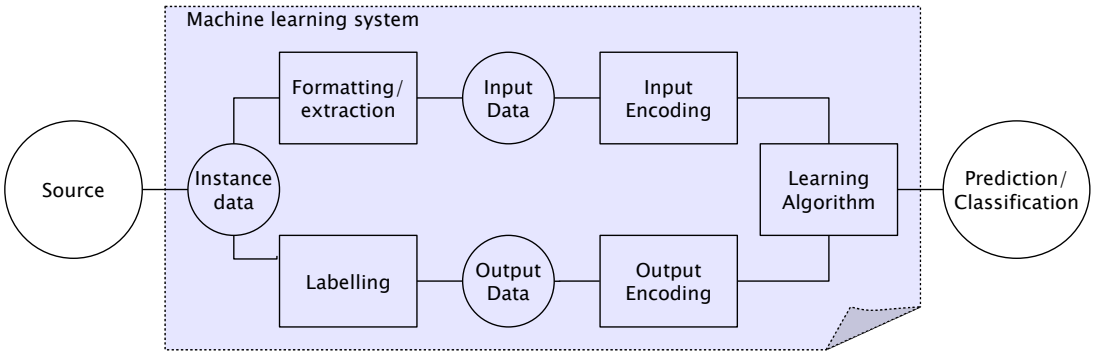


Figure 4.1: A generic prediction problem decomposition.

data types. The actors are:

- *Input Data* – represents an instance of data to be predicted. No limits are given to its definition: for instance, it could be a vector, a text, a pointer to a file, a protein primary structure, a signal or an image.

- *Output Data* – represents an instance of *what* we are predicting, and has a twofold value; in the training phase, it represents the target data. In the prediction phase, it represents the predicted data.

- *Instance Data* – defines an instance of data in terms of input/output, the loading format, the iteration and evaluation policies. An Instance Data contains an Input Data and Output Data. The type of Instance Data is a general parameter, unique for the whole system.

- *Input Encoder* – Encoders are the interface between data and learning algorithms. Input Encoders apply a transformation on an input data, returning an *Encoding* object. An Encoding is a stream of training instances, i.e. normalized vectors of fixed size directly usable to feed a learning algorithm. For atomic data, the input encoding applies a single transformation on the target data. For structured

data, an Encoding typically wraps a matrix in which the the different training instances can be extracted with sliding windows or other mechanisms.

- *Output Encoder* – applies a transformation on an output data, returning an Encoding. Output Encoders apply a reversible transformation, so that the original output data which originated an encoded instance can be obtained from it.

Some encoders may be data-specific, others may be suitable for classes of data. For instance, nominal data can always be managed by one-hot transformation, whereas PCA transformations are quite a general way to reduce a vectorial input space. The decomposition proposed in GAME promotes the reuse of learning methods, Encoders techniques and data types, with the development of hierarchies in the same object-oriented framework.

A *Predictor* is an object able to return a (predicted) Output Data from an Input Data. It is composed by an *Expert* and a *Decoder*. Experts are the central computing element in GAME; they give a prediction from an Input Data. Predictions have an encoded form, which can be reverted back to the human-interpretable data which originated it with the help of a Decoder (see Chapter 6 for an example of output encoding/decoding for secondary structure prediction).

A *Learner* is an Expert with learning capabilities. A base Learner contains a *Learning Algorithm* (a generic wrapper of a learning algorithm) an Input Encoder, an Output Encoder. These modules can be chosen from the palette of available modules for the actual Instance Data. A Learner can be trained by a *Meta Learner*, which may build a single learner or an ensemble of learners. A Learning Algorithm knows how to train itself from a training set, in the form of *Dataset Iterator*. A Dataset Iterator gives the training instances at request, abstracting from the encoding phase. Dataset Iterators can be constructed from *Datasets*, which represent collection of instances, by specifying the Input and Output Encoders to be applied to every instance.

4.1.2 Expert Interaction

A notable characteristic of GAME is the native support for handling the interaction among software experts. Multiple experts have been mainly adopted in the evolutionary-computation and in the connectionist communities. In the former, the focus is on devising suitable architectures and techniques able to enforce an adaptive

behavior on a population of individuals, e.g., Genetic Algorithms (GAs) (147), Learning Classifier Systems (LCSs) (148), and eXtended Classifier Systems (XCSs) (149). In the latter, the focus is mainly on training techniques and output combination mechanisms; in particular, let us recall the pioneer work of Jordan's Mixtures of Experts (150) and Weigend's Gated Experts (151). Further investigations are focused on the behavior of a population of multiple (heterogeneous) experts with respect to a single expert (152). Expert interaction is realized through *has-a* relationships. In particular, an expert may belong to one of the following categories:

- *Ground Expert:* A ground expert is an independent expert, able to output its classification or prediction without resorting to any other expert. Among the ground experts, let us particularly recall (i) Learners, (ii) Wrappers. Learners, which represent the core of GAME, are concerned with the adoption of machine learning techniques or strategies. Available supervised techniques include MLPs and Bayes classifiers. Principal component analysis is also available among unsupervised techniques. Wrappers allow the embedding of external classifiers or predictors, including available web services or external programs that one wants to use. They also allow the implementation of specific (hand-crafted) behaviors.

- *Refiner:* Refiners are the technological solution adopted in GAME for implementing sequential combinations of experts (i.e., pipelines). A Refiner is an Expert that can embed another Expert. A pipeline of Experts can be easily created by repeatedly applying refinement. Once generated, a pipeline becomes a compound Expert entrusted with processing available data that flow along the pipeline. Each Expert in the pipeline, while processing information output by the previous expert, can be optionally supplied with additional information.

- *Combiner:* Combiners are the technological solution adopted in GAME for implementing parallel combinations of experts. A combiner is an Expert that can embed other Experts. A custom ensemble of Experts can be easily created using a Combiner. Once generated, an ensemble can be considered a compound Expert in which the Combiner collects and processes the output of other experts.The following output combination policies have been implemented so far:

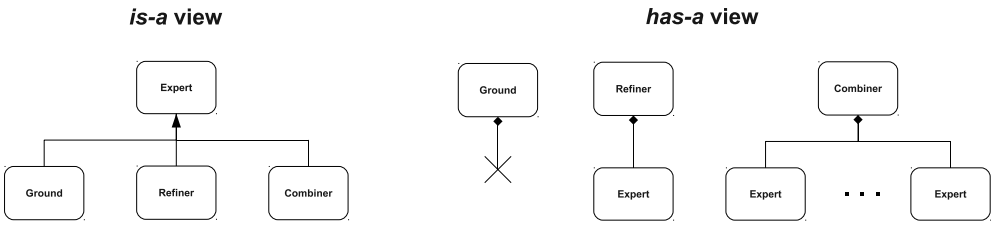


Figure 4.2: GAME experts: *is-a* and *has-a* view

- *Averaging* (AVG). The prediction is given by averaging over the set of involved experts (separately for each residue).
- *Weighted-Averaging* (WAVG). A variant of averaging, where the contribution of each expert is measured in accordance with the reliability enveloped in the prediction.
- *Elite Averaging* (EA). A variant of averaging, where only a given percentage of experts are allowed to participate to averaging. Experts are ranked according to their reliability and selected for averaging until the given percentage is reached.
- *Majority Voting* (MAJ). Each expert issues a single vote and then the most successful label is selected.
- *Elite Majority Voting* (EM). A variant of majority voting, where only experts with a reliability that overwhelm a given percentage are allowed
- *Stacking* (STK). A learner is trained on the outputs provided by the set of involved experts upon several training inputs.

Mixture of Experts can be also realized, thanks to the possibility to specify gating functions to individual experts together with a combination policy.

4.2 GAME Framework: Implementation Details and Standard Modules

The GAME software is completely written in Java 6.0[1] with the integrated development environment NetBeans[2]. From a design-oriented perspective, the "divide et impera",

[1] `http://www.java.com`
[2] `http://netbeans.org`

the object-oriented programming polymorphism and code refactoring (153) principles have been widely applied. The Model View Controller pattern (154) has been applied at different levels to separate the logic of the system from its configuration. System logic is based on the interaction of a hierarchy of system interfaces that represent pluggable modules. In order to guarantee the maximum independence between modules, only system interfaces and standard types are allowed as parameters of exported operations.

A module in GAME is a Java class in a specific package which implements a specific interface. Implementing a new module, with the proper use of inheritance from the standard modules available, usually requires the implementation of just a few methods.

Hierarchies of standard modules provide support for configuration and implement the main low-level operations and commonly used operations. These modules deal with configuration with automatic field inspection and annotations, and can be configured by graphical interface or by XML descriptors. XML descriptors, or alternatively the Java serialization API, permit to store and retrieve the system configuration and modules from run to run. Both the possibilities are given because each type of serialization has its good points and drawbacks. While the serialization API generates compact and fast object representations, it is more sensitive to changes in the implementation than a more abstract descriptor. XML descriptors give also the possibility to edit the configuration manually, so giving the possibility to rapidly diversify experiments with text editors' search/replace or copy/paste features.

The GUI is the easiest way to generate configurations. It is implemented using the standard Java *Swing* lightweight library and it is based on a mixture of ad-hoc and automatically generated configuration windows.

4.2.1 Defining and Running Experiments

The most important use of GAME is defining experiments. An Experiment is a module which is configured by choosing and configuring other modules. The configuration can be carried out by editing an XML file or, more easily, with the graphical user interface provided. Different modules contribute to defining an Experiment, the most important being the type of Instance Data considered and the main Predictor. Figure 4.3 shows the modules involved in the configuration of an Experiment.

In GAME, Experiments are runnable modules which define the computation flow. An experiment generally initializes and tests a *Predictor* according to its configuration.

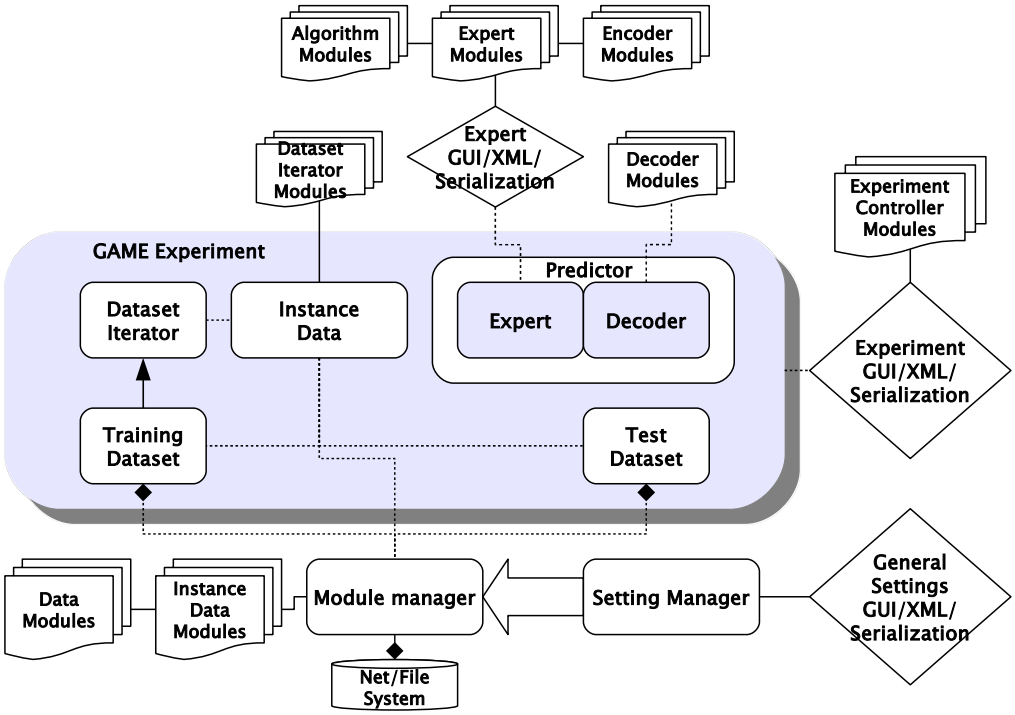


Figure 4.3: Experiment definition. A dotted line stands for the instantiation and/or configuration of a module, a 'multiple sheet' box indicates a palette of alternative modules to choose from.

A *Settings Manager* is devoted to the configuration of the type of data and general settings, such as logging, caching, dataset resource files. A Module Manager, designed in accordance with the Factory design pattern, is responsible for system module handling. Figure 4.4 shows the main experiment configuration GUI. Different experiments can be configured and launched in the same session.

4.2.2 Defining GAME modules

All modules in GAME have some common properties, which permit them to be configured and serialized in the same way. All modules must implement the standard Java interfaces `Cloneable` and then provide a public `clone` method, with deep copy implementation. They must also implement `Serializable`, in order to be saved and reloaded with the standard Java serialization API. Other custom interfaces required to GAME modules are:

- `Configurable`. Defines the configuration with a Map-like interface, with

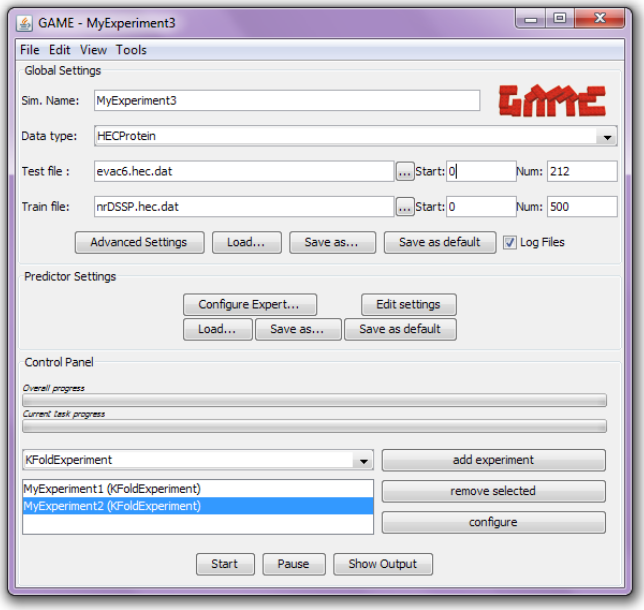


Figure 4.4: The main configuration window of GAME. It permits to define and run experiments.

key-value properties. The method `getOptions` gives the map of key-values, `getOption` returns a value for a key, `setOptions` and `setOption` allow to modify the configuration.

- `XMLConfigurable`. Defines `getXMLSettings`, which returns an XML descriptor of the configuration, and `loadSettingsFromXML`, which can load the configuration from an XML descriptor. In addition, methods for default configuration loading and saving are defined.

- `XMLLoadable`. Defines methods for XML serialization.

Although configuration and serialization may seem redundant definitions at first they are two very distinct concepts: configuration is thought to be user-friendly and defines the initial state of an object (for example, before a training process), while serialization can define any state.

Different implementation may be given which satisfy all the listed requirements, the simple one being to inherit from the class `GAMEModule`. Almost all the standard

modules inherit, directly or indirectly, from it[1].

A `GAMEModule` object considers all its public fields as part of the configuration (i.e. values of the map returned by `getOptions`), and implements all the constraints cited above without the need of any further configuration. All the standard types and modules are handled through automatic introspection of the fields. XML configuration is implemented with a recursive approach, while XML serialization uses the external library *XStream*[2]. The method `clone` is implemented resorting to Java serialization. The method `getDocumentation` automatically builds the documentation for the module. The method `getClassDescription` can be overridden to specify a general description for a module, while the annotation `Description` on a parameter field permits to add a documentation for that parameter.

Another way to provide the documentation, together with safe validation controls, to a configuration value is to define it as a *Value Holder*. Value holders are wrappers of simple and more complex types which wrap a value. The value can be set only with a specific method, which can perform all the validation controls needed.

A further requirement which is requested to some modules is to be inside a specific package; in particular, modules which could be substituted each other should be in the same package. Standard packages are fixed for the standard modules, but one could define her hierarchy of replaceable modules and choose the package for that modules at her own will. If all the requirement are followed when implementing a module, the graphical user interface of GAME includes it in the list of selectable modules, permits to read the documentation provided and automatically generates a configuration panel, with facilities to load and save the configuration for that module.

Let us see an example code:

```
public class MyModule extends GAMEModule {
    @Description(description="My first parameter")
    public int par1 = 0;
    public boolean par2 = true;
    public InstanceData par3 = new CSVData();
    private double hiddenState = 0;
```

[1] In most of the rest of the document the inheritance from `GAMEModule` and the implementation of the cited interfaces is not indicated for the sake of clarity.

[2] `http://xstream.codehaus.org/`.

```
    @Override
    protected String getClassDescription() {
        return "My first module";
    }
}
```

The above code is all it's needed to create a documented module named *MyModule*, with three parameters *par1*, *par2*, *par3*, and with an additional variable defining the state. The framework automatically creates a configuration and a documentation window for that module, both displayed in Figure 4.5. That window is opened auto-

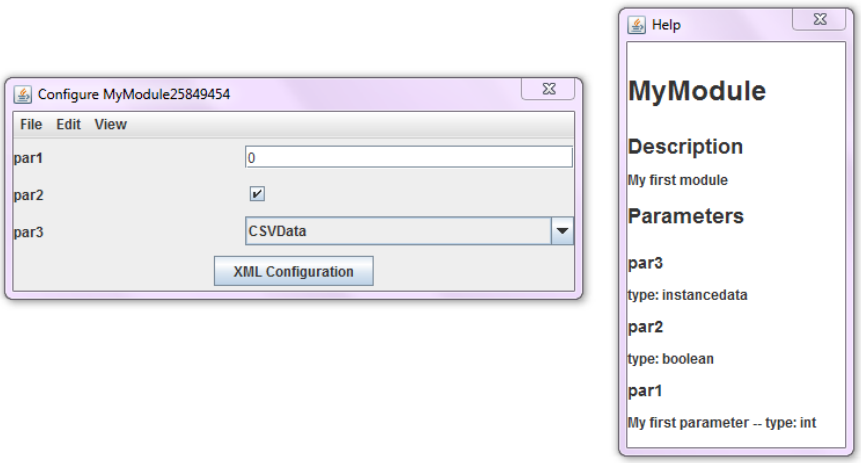


Figure 4.5: Screenshot of an automatically generated configuration and documentation windows for a GAME module.

matically at need if the module is part of the configuration of a module recognized by the system; otherwise, it can be displayed with the statement `new OptionsDialog(new MyModule()).setVisible(true)`. From the *menu* file of the configuration window is possible to save and load the state and the configuration. The XML configuration descriptor for *MyModule* is:

```
<mymodule class="game.MyModule" >
    <par1>0</par1>
    <par2>true</par2>
    <par3>
        <csvdata class="game.datahandling.dataenvelopes.CSVData" >
       </csvdata>
```

```
    </par3>
</mymodule>
```

While the XML configuration descriptor is:

```
<object-stream>
  <game.MyModule>
    <par1>0</par1>
    <par2>true</par2>
    <par3 class="game.datahandling.dataenvelopes.CSVData">
      <id>NOT-INITIALIZED</id>
      <input class="game.datahandling.dataenvelopes.base.VectorData">
        <encoder>
          <size>-1</size>
        </encoder>
        <value class="double-array"/>
      </input>
      <target class="game.datahandling.dataenvelopes.base.VectorData">
        <encoder>
          <size>-1</size>
        </encoder>
        <value class="double-array"/>
      </target>
    </par3>
    <hiddenState>0.0</hiddenState>
  </game.MyModule>
</object-stream>
```

4.2.3 Standard Modules for Prediction an Learning

Defining modules related to prediction and learning is the most important use of the GAME framework. This section is a reference for the user of the framework who wants to start the study of a new prediction problem.

As shown in Figure 4.6, each module described in Section 4.1.1 is defined by an interface in the framework. A hierarchy of interfaces and subclasses starts from each interface. Classes which implement these interfaces, located in the proper sub-package of the package of the interface, constitute the plugins that are automatically handled

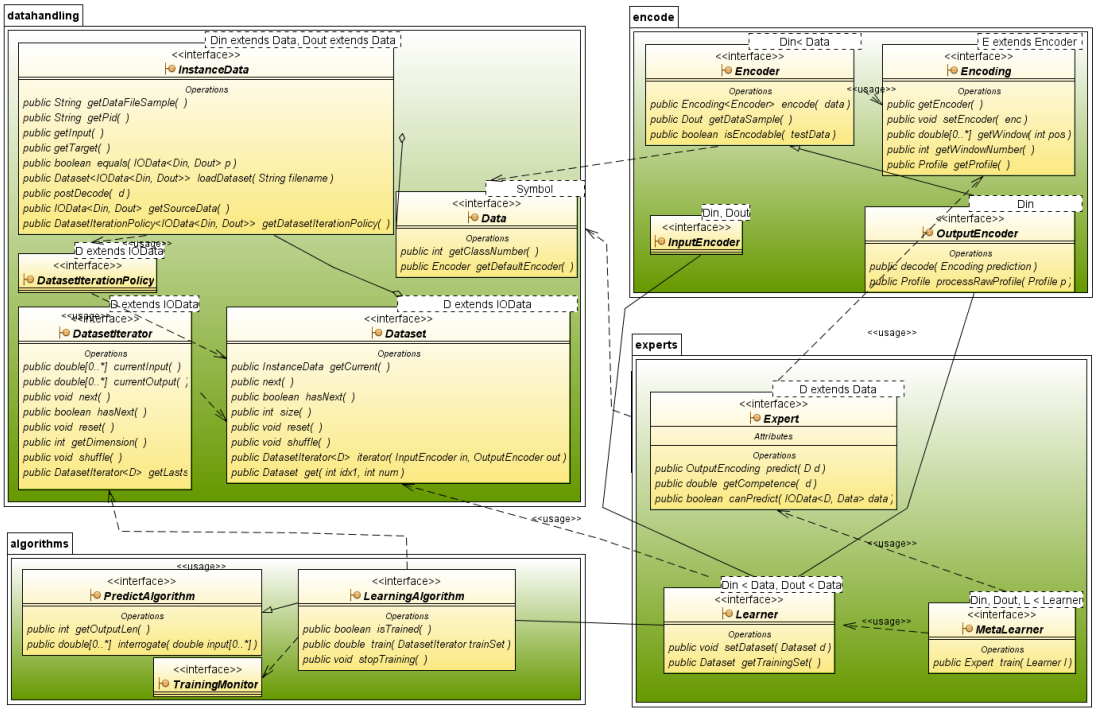


Figure 4.6: Base GAME prediction and learning system modules: UML class diagram (built by code reverse engineering with UML plugin for NetBeans 6.7).

by the framework. The user who wants to create a new module must implement one of those interfaces, usually extending one of the existing general implementation, and putting it in the proper package.

The main task in the definition of a new problem is the definition of a module for the Instance Data. This is done by implementing the interface *InstanceData*. An Instance Data contains two instances of `Data`, representing the input and the target. Different off-the-shelf general Data modules are available to choose which can be directly used or extended to implement distinctive features of the problem. In a typical case, a new type of input Data must be defined for a new problem, while one of the available modules can be generally found which properly adapts to the target data.

Experimenting different ways to encode data by defining different input and output Encoders is the best way to exploit GAME. Encoders are parametrized for a Data type or a hierarchy of Data types, and return an instance of `Encoding`. It is not usually required to define a new type of Encoding, as well as it is not required to define a new Dataset or Dataset Iterator.

Standard Experts and Learners are generally used as they are. Defining a new Expert may be useful to implement an ad-hoc algorithm (for example a naive algorithm to be compared with other techniques), to experiment new post-processing or combination techniques between experts, or to wrap external programs or web services.

Ad-hoc algorithms can also be realized by implementing the simple interface `PredictAlgorithm`. Implementing the interface `LearningAlgorithm` permits to add a new Learning Algorithm to the system.

In the following sections, all the interfaces introduced above are explained, and the standard general modules available for each one are described.

4.2.3.1 Instance Data

Defining an Instance Data requires the implementation of the interface `InstanceData`, which main methods are[1]:

- `getInput` – returns the input Data for the instance. The straightforward implementation is a getter of a inputData attribute; more advanced implementations can generate the data instance just-in-time, for example loading it from an external resource or compute it to generate synthetic data, or load a random Data from a set of similar Data.

- `getTarget` – returns the target Data for the instance. The same observations of `getInput` are valid for *getTarget*.

- `loadDataset` – returns a Dataset for the current Instance Data type, loading data by parsing a file. It is semantically a class method, but it is implemented as an instance method due to the limitations of the Java language, which does not allow to specify class methods.

- `getDatasetIterationPolicy` – returns a DatasetIterationPolicy, a factory of DatasetIterators. Changing the iteration policy is the best way to handle personalized sampling of data instances or, for sequential data, the order of window extraction.

[1] Implementation of other methods than the ones indicated can be required; standard Java methods such as `equals`, `hashCode`, `toString` and `clone`, as well as methods whose implementation is straightforward or not necessary are not indicated here and in the next sections for the sake of brevity.

- `getEvaluator` – returns the evaluation module to be used to test the Predictor, as an instance of `Evaluator` (see Section 4.2.5 for more information).

Two main types of Instance Data are available as abstract classes, which define `getInput` and `getTarget` with two attributes for input and target Data, and give a default implementation of `getDatasetIterationPolicy`:

- `AtomicInstanceData` – uses by default a static sequential dataset iteration policy (`FastAtomicDataDatasetIterator`, see Section 4.2.3.7).
- `StructuredInstanceData` – uses by default a sequential iteration over instances and windows (`StructuredDataDatasetIterator`, see Section 4.2.3.7).

Instance Data modules are required to be located inside the package `game.datahandling.dataenvelopes` in order to be recognized from the system A general-purpose implementation of Instance Data is `CSVData`. It reads data from a CSV file and interpret the values as vectors of floating points (*VectorData*, see Section refsec:data).

4.2.3.2 Base Data

Data types are statically defined within the main Instance Data type. Although they are not pluggable modules, their definition is not different from other standard modules, except the fact that they do not require any save/load machinery. Data types are described by the interface `Data`, which requires the implementation of the following methods:

- *getClassNumber*. The number of possible different values for that data. The value 1 is interpreted as if infinite values are allowed. Knowing the number of possible classes allows to use general purpose encoding methods for multiclass data.
- *getDefaultEncoder*. Returns an instance if the default Encoder for that data.

Many kinds of Data are available off-the-shelf. The most important ones are shown in Figure 4.7. At the first level, four different types of Data are defined:

- `AtomicData`. Represents a simple data embedded in an `AtomicInstanceData`.

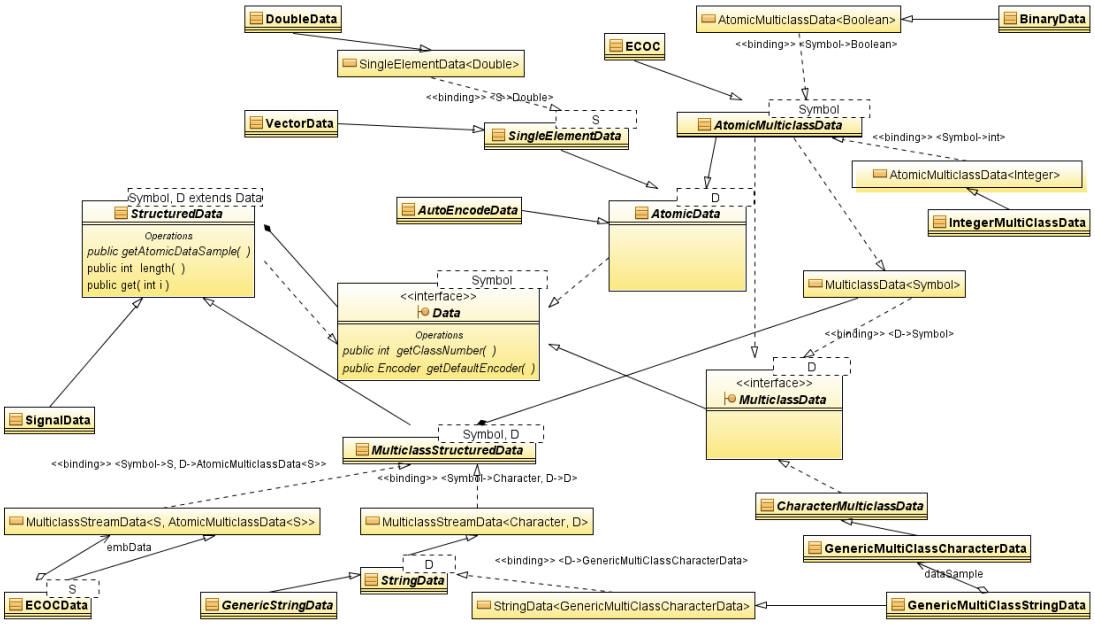


Figure 4.7: Hierarchy of data types in UML.

- `StructuredData`. Represents data embedded in an `StructuredInstanceData`. Structured Data can be expressed as a sequence of other data. The method `length` returns the number of element contained, and the method `get` returns a data element at a specified position.

- `MultiClassData`. Represents data whose class number is defined. Atomic and Structured Data can also be multi-class.

The cited types are specialized according to the type of data which they represent.

`SingleElementData` embed a single –non multi-class– value, such as a floating point (`FPData`) or a vector (`VectorData`).

Common types of multi-class data are integers (`IntegerMultiClassData`), booleans (`BinaryData`), characters (`CharacterMulticlassData`).

Structured Data are distinguished according to the type of data contained. For example, `StringData` contain `CharacterMulticlassData`, while `SignalData` contain `VectorData`.

`AutoEncodeData` represent data which embeds multiple values. Its subclasses can express auto-encoding values by specifying fields marked with the annotation `AutoEncode`.

4.2.3.3 Encoders and Encodings

Defining Encoders is a common operation while working with the GAME framework. Encoders can be defined by implementing the interface `Encoder`, which is parametrized with the Data which encodes, `Din`. The main methods of the interface `Encoder` are:

- `encode`: returns an instance of `Encoding` for the Data given in input.
- `isEncodable`: returns `true` if the data given as input can by dealt with this Encoder. In practice, it verifies whether the type of the parameters is subtype of `Din`[1].

in the definition of a new module, the interface `Encoder` is never implemented directly; one of the derived interfaces is implemented instead, according to the role of the Encoder in the System.

- `InputEncoder`: a marker interface, indicating an Input Encoder. An Input Encoder is not require to return a reversible Encoding. The automatic configuration system automatically recognizes Input Encoders located in the package `game.encode.input`.
- `OutputEncoder`: indicates an Output Encoder, which returns a reversible Encoding and defines the method `decode`. The method `decode` is able to reverse the method `encode` and it is also able to decode an Expert's output. New Output Encoders should be put inside the package `game.encode.output`. Then, the simplest Decoder is a module that just calls *decode* on Expert's output.
- `RefinementEncoder`: used by Refiner Experts to implement pipelines. A Refiner Encoder may apply some transformation to the Encoding produced by the Expert embedded in the Refiner. Refinement Encoders are expected to be located inside the package `game.encode.refinement`.

Some Encoders can be suited both for input and output; in this case, they should implement both `InputEncoder` and `OutputEncoder`. This input/output encoders are located in the package `game.encode.generic`.

Figure 4.8 shows the standard hierarchy of Encoders. Being Encoders and Data

[1] An explicit implementation is required as soon the Java language does not provide a way to control the parametrized type of an object at runtime.

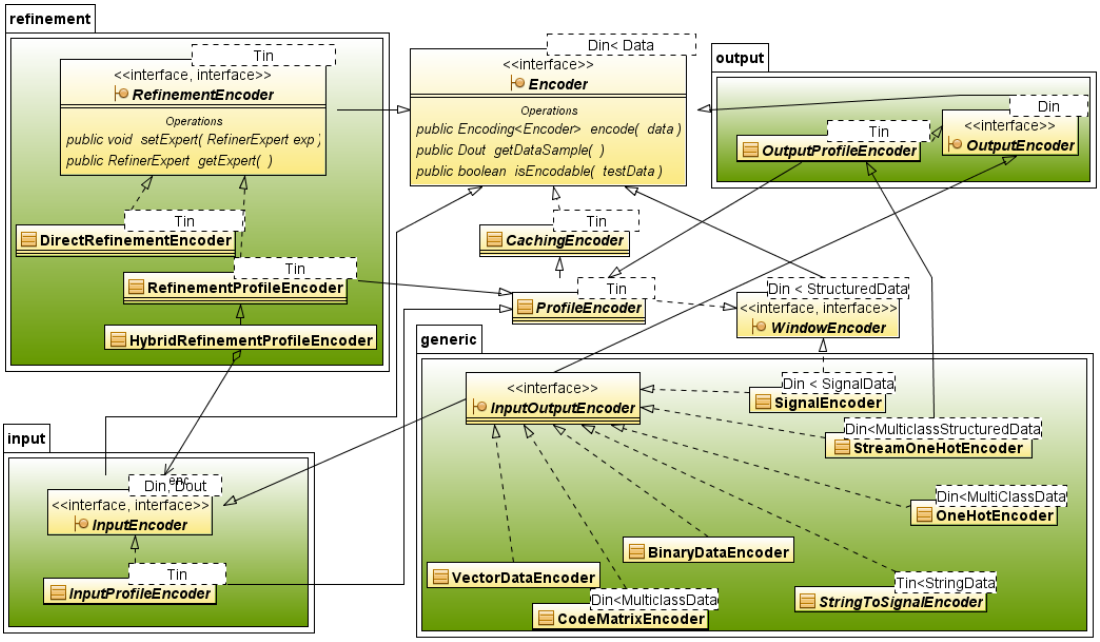


Figure 4.8: Hierarchy of base Encoders in UML.

types bound one each other, the hierarchy of Encoders partially resembles the hierarchy of Data.

Multi-class data can be always dealt with a `CodeMatrixEncoder`. Code matrix is a general reversible method to associate a vector data with a limited number of instances. It can be used to implement simple binary codes or more complex error-correcting output codes, and decoding is performed with a minimum Hamming distance criterion.

One-hot coding (`OneHotEncoder` and `StreamOneHotEncoder`) is a widely special case of code matrix coding: a binary vector, with length equal to the number of classes, is used, and the position i the vector is high to indicate the ith class.

Encoders for Structured Data are based on Profiles, which are matrices in which each column is related to a Data sub-element. They are implemented with the module `ProfileEncoder`. A Profile Encoder produces a `WindowProfileEncoding`, which extracts sliding windows from the profile (Figure 4.9). The size of the window is specified by the parameter *windowSize*.

A sliding window for position j of size n_w includes all profile columns in the range $[j - \frac{n_w}{2}, j + \frac{n_w}{2}]$ For the positions out from the profile an additional input can be added

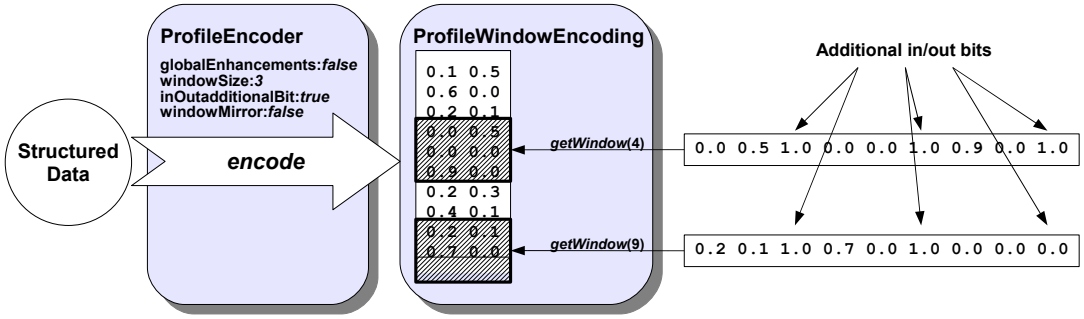


Figure 4.9: Example sliding window extraction.

(parameter *inOutPositionAdditionalBit*) to indicate whether the position is inside or out the profile. As for the values, two alternatives are given – (i) set all the out of bounds elements to zero, (ii) mirror values, i.e. taking $j + \frac{n_w}{2}$ when $j - \frac{n_w}{2}$ is out and vice-versa.

Global values for the full profile can be appended to each window enabling the parameter *globalEnhancements* and specifying the method `initializeGlobalInfo`.

`ProfileEncoder` derives from `CachingEncoder`, which implements a mechanism to cache the Encodings when calculated the first time. Caching is based on dictionaries of soft references, in order to automatically be safe from memory leaks.

A separate attention deserve Refinement Encoders. `DirectRefinementEncoder` is a proxy of the embedded Expert's output and is suited for all kinds of data. `RefinementProfileEncoder` can be used for Structured Data when the Output Encoder of the embedded Expert is a Profile Encoder. A new Profile is constructed based on the Profile of the Expert's output, and new global values can be computed and added to the windows. `HybridRefinementProfileEncoder` appends an input Encoding (produced by an `InputProfileEncoder` selected among the available modules) to the embedded Expert's output profile.

The interface `Encoding` embeds the result of the encoding process performed by an Encoder. The main method is `getWindow`, which gives an encoded instance as fixed-size array of floating point numbers, and it is called to get the training instances by Dataset Iterators. The method `getWindowNumber` specifies the number of windows that can be extracted from an Encoding. Typically, the number is 1 for Atomic Data, and coincides with the value returned by the method `length` for Structured Data.

Every Encoding knows the Encoder which produced it; Figure 4.10 shows the standard Encoding classes hierarchy. `ArrayEncoding` and `SingleWindowProfileEncoding`

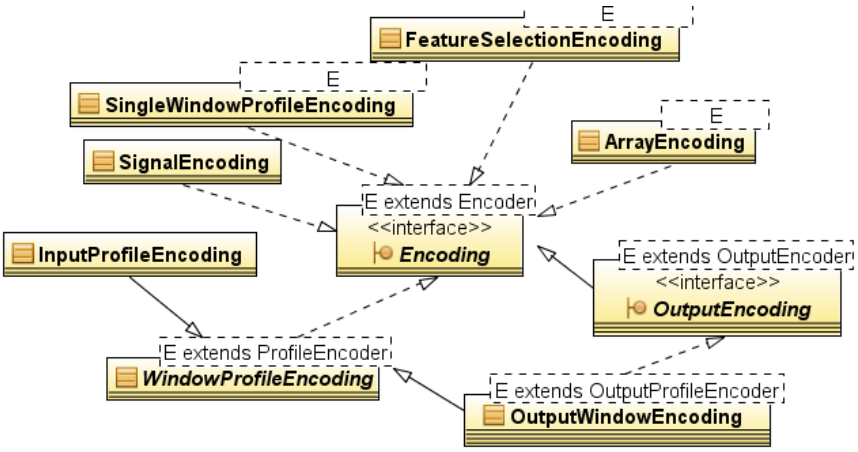


Figure 4.10: UML class diagram of standard Encoding modules.

are normally used for Atomic Data, whereas `WindowProfileEncoding` is used for Structured Data encoded with a `ProfileEncoder`.

4.2.3.4 Decoders

Decoder modules are located inside the package `game.decoders` described by the interface `PredictionDecoder`. Since Output Encoders are able to decode their output, the standard module which applies that conversion (`SimpleDecoder`) is suited for all properly defined Output Encoders. Further Decoders should be implemented where custom policies or post-processing operations are desired.

4.2.3.5 Experts

Defining Experts is one of the central purposes of GAME, and facilities to define, train, test, configure, save and load Experts are all provided by the graphical interface. On the framework side, defining a new type of Expert is not the kind of operation that is expected to be done frequently while defining a new problem with GAME. New Experts may be defined to implement wrappers, ad-hoc prediction or combination techniques.

The interface `Expert` declares three main operations:

- `predict`: returns an instance of `OutputEncoding`, which embeds the result of the prediction.

An Output Encoding is returned instead of the predicted data directly to use it for further pipelines and combination and avoid the any unnecessary loss of information caused by a decoding process.

The method `predict` may return a `null` value in the case that the Expert is not expected to predict anything.

- `getCompetence`: returns a value in the range $[0, 1]$, meaning the relative competence of the Expert for the Data given.

  The value can be used in combinations, and may be a static value derived from the knowledge of the competence of the technique applied.

  Competence may be obtained with rules deriving from partition of the Data or may simply be the performance of a validation set used during a training phase.

- `canPredict`: says whether the given Data can be predicted or not by this Expert.

The three main types of Expert, as described in Section 4.1.2, are represented by the interfaces `GroundExpert`, `RefinerExpert`, `CombinerExpert`. Every Expert is expected to implement one of those three interfaces. An Expert can also be a Learner (interface `Learner`), or a Wrapper (interface `WrapperExpert`), which is a term to indicate an independent ground Expert.

A Wrapper could "wrap" a prediction system running outside GAME, or any ad-hoc implementation. The user of the framework can define new Wrappers inside the package `game.experts.ground.wrappers`, and the GUI gives the possibility to select and configure the defined Wrappers.

The user could also define and use its own Combiners by implementing the interface `CombinerExpert` inside the package `game.experts.combine`. As well as other modules, implementation of Combiners is made easier a hierarchy of general-purpose Experts.

Figure 4.11 shows a view of the main Expert hierarchy (Learners are excluded here for the sake of clarity).

The most important implementation of Expert is `EncodingBaseExpert`, which follows the generic decomposition shown in Section 4.1.1, and implements a Gating Function, which defines the domain of competence of the Expert. A Gating Function can be specified by configuring a suitable module subclass of `GatingFunction` (see Section 4.2.3.8). `AlgorithmExpert` allows to use generic and ad-hoc modules for prediction

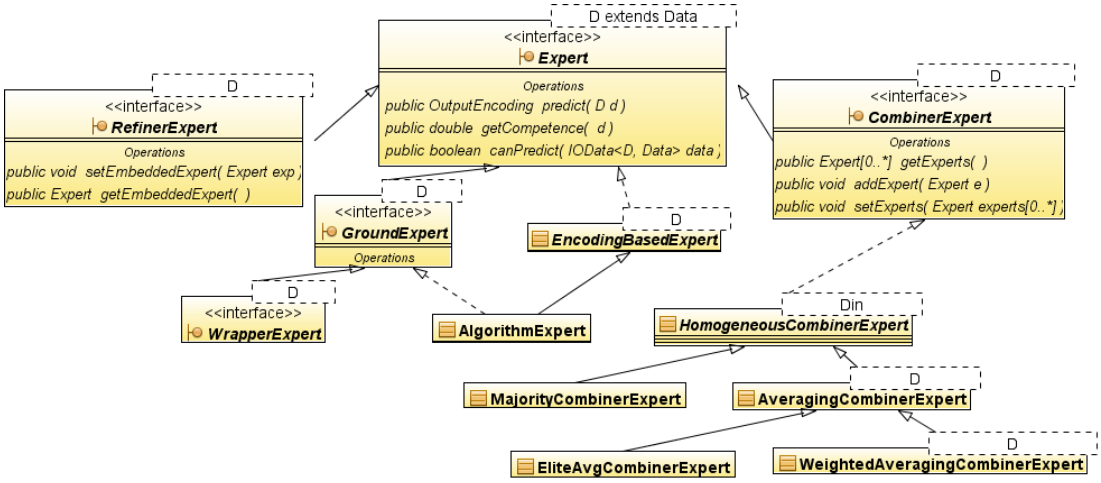


Figure 4.11: UML class diagram of the main hierarchy of Experts (Learners excluded).

which implement the interface `PredictAlgorithm` (Section 4.2.3.6), while using the input Encoder modules.

As for Learners, two different hierarchies are defined in GAME, one simpler to configure (Figure 4.12) and one more flexible thanks to the use of Meta Learners (Figure 4.13). `LearningExpert` and `LearnerExpert` are the main implementations of the

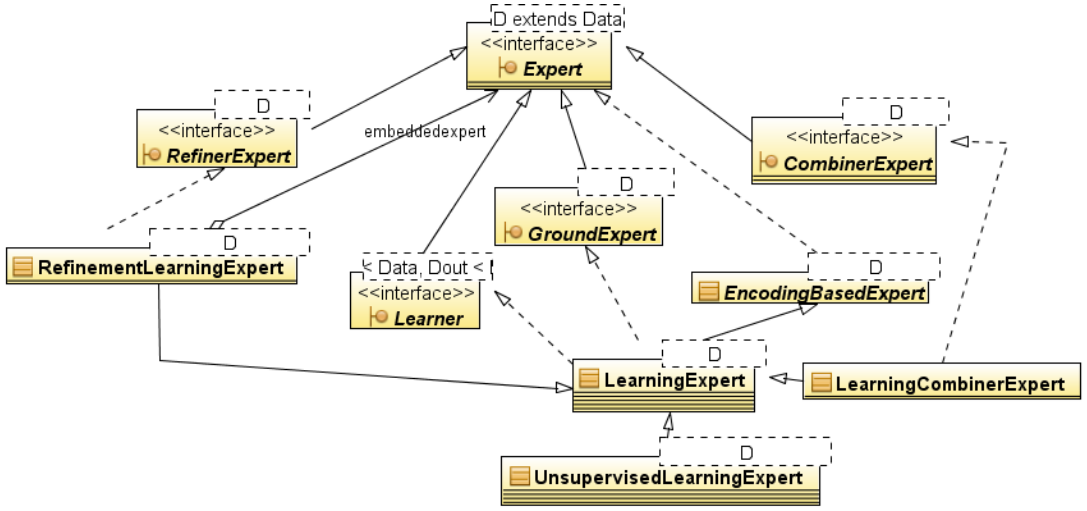


Figure 4.12: UML class diagram of the basic Learner modules.

interface `Learner`. Both use an input Encoder, an output Encoder and a Learning Algorithm, which can be selected and configured separately. The only practical difference

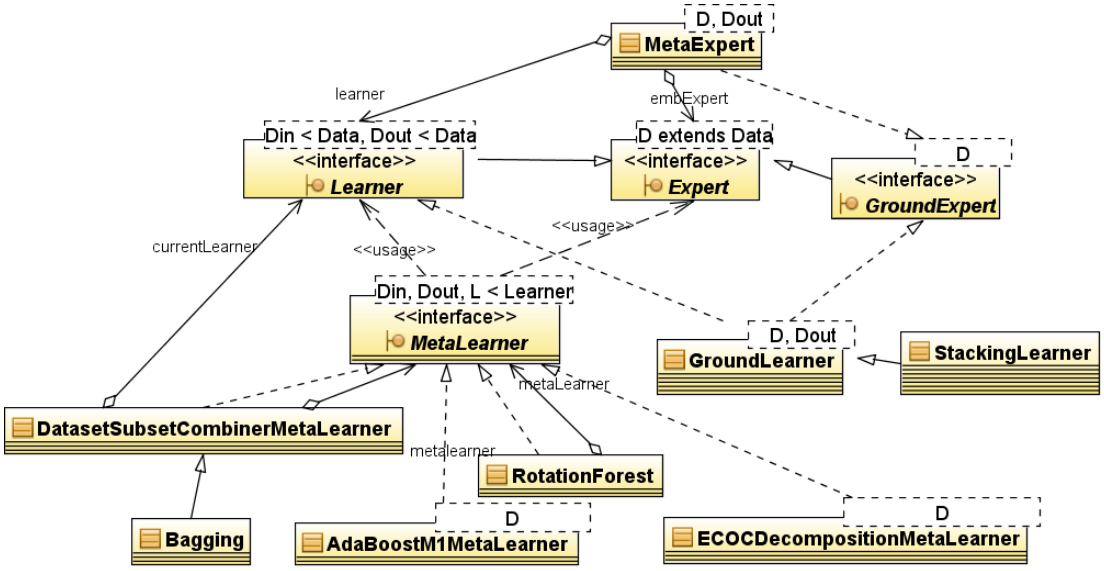


Figure 4.13: The meta-learning hierarchy of Experts (UML class diagram).

between the two is that the second needs a Meta Learner to complete its training.

Available Meta Learners include implementations of stacking(155), bagging(140), boosting(141), rotation forests(156), ECOC decomposition (142). The class `MetaExpert` handles all the mechanisms behind meta-learning, which is the application of a Meta Learner to a Learner that produces an Expert. It is worth noting that although most meta-learning techniques produce ensembles, the result of their application does not produce a Combiner. This choice has been made because the prediction units produced by meta-learners are in most of cases not independent: their proper combination is part of the meta-learning algorithm.

4.2.3.6 Prediction Algorithms

A Prediction Algorithm applies a transformation $\mathbb{R}^{n_i} \rightarrow \mathbb{R}^{n_o}$, where n_i and n_o represent the dimension of the input and output Encoding window, respectively. The abstraction provided by the input and output encoders allows to define problem-independent algorithms.

Prediction Algorithms are modules which implement the interface `PredictAlgorithm` and are located within the package `game.algorithms`. The interface is maintained really simple, the only important method to be implemented

being `interrogate`, which takes as input and output parameters simple arrays of floating point numbers. No any knowledge of the framework is needed to implement a Prediction Algorithm for GAME. As shown in Figure 4.14, there are *processing* and *learning* algorithm modules. General purpose processing algorithms are feature

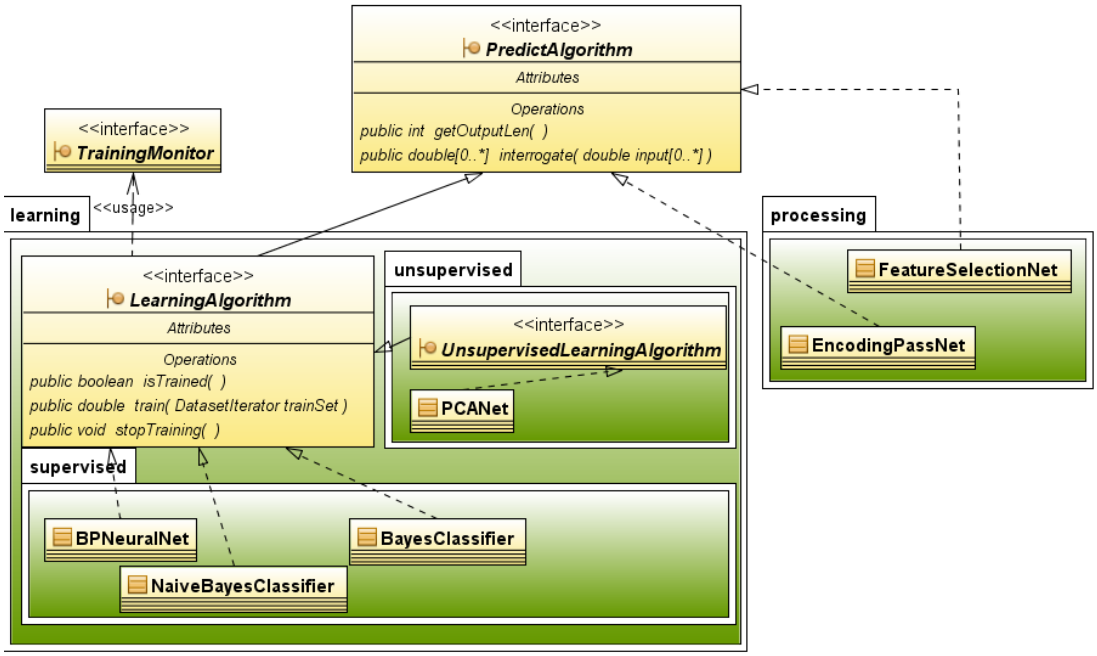


Figure 4.14: Hierarchy of the main generic algorithms in UML.

selection (`FeatureSelectionNet`) and Fourier transforms (not available yet).

Supervised and unsupervised are the two different categories of Learning Algorithms. Supervised algorithms are multi-layer perceptrons (an implementation being `BPNeuralNetwork`) and (naive) Bayes classifiers (modules `BayesClassifiers` and `NaiveBayesClassifier`). Unsupervised techniques include clustering (not available yet) and principal-component analysis ((157), module `PCANet`). All learning algorithms are trained resorting to a Dataset Iterator, whose implementation details can be completely ignored at this level.

Developing many different algorithms has not been been a priority in the development of the GAME; the available algorithms cannot be compared with the wide choice provided by tools like WEKA and RAPIDMINER. In any case, a wrapper of the WEKA algorithms could be easily provided. On the other hand, using the algorithms from WEKA would suffer of the same memory problems with big training

sets; directly using WEKA may be a better choice where a wide comparison between learning algorithms is wanted. An export utility is provided from the main GUI menu allows to export a Dataset Iterator to a CSV file, choosing the desired input and output Encoders.

4.2.3.7 Datasets

Datasets are needed to train and test predictors. Two dataset levels are considered in GAME, in order to handle the decomposition of structured data. The first level, represented by the interface `Dataset`, is a collection of Instance Data, regardless of its specific type. The second level is represented by the interface `DatasetIterator`, which wraps a Dataset iterating on the single instances obtained after the encoding phase. Dataset Iterators are using during a training process, while Datasets are directly used for testing purposes.

The interface `Dataset` extends the standard Java interfaces `Iterable` and `Iterator`, parametrized for Instance Data. Other exported methods are:

- `getCurrent` – returns the current Instance Data.
- `reset` – resets to the initial state; the first Instance will be given by `getCurrent`.
- `shuffle` – randomly changes the order of iteration and resets the Dataset.
- `join` – merges the content with another dataset, avoiding duplications.
- `size` – returns the number of elements of the Dataset.
- `getFirsts` – gives a new dataset with first percent elements. For instance, `getFirsts(0.5)` returns a dataset on the first half elements.
- `getLasts` – the twin of `getFirsts` for the last elements.
- `iterator` – given input and output Encoders, returns a Dataset Iterator over itself.

The standard implementation of `Dataset` is `ArrayDataset`, which uses an array to store the elements contained.

The interface `DatasetIterator` resembles the interface of Dataset. Being a Dataset Iterator a wrapper of a Dataset, the method `reset`, `shuffle`, `getFirsts`, `getLasts`

should be delegated to the embedded Dataset. The standard iteration methods `hasNext` and `next` are also defined, with the main difference that `next` goes to the next element of the iteration without returning it, to improve efficiency. The distinctive methods of `DatasetIterator` are:

- `getInput` – returns the current input as a floating point array.
- `getOutput` – returns the current input as a floating point array.
- `getDimension` – returns the number of elements of an iteration, i.e. the number of times that `next` can be called after a reset before `hasNext` becomes *false*. The size of an iteration is constant, but may not be related to the number of elements of the embedded Dataset.

An example use of Dataset Iterators to train a Learning Algorithm is:

```
class MyAlgorithm extends GAMEModule implements LearningAlgorithm{
    ...

    public double train(DatasetIterator d) {
        for (d.reset(); d.hasNext(); d.next()) {
            double [] predOutput = this.interrogate(d.currentInput());
            double [] obsOutput = d.currentOutput();
            ... //training step
        }
    }
}
```

Dataset Iterators can be composed following the *Decorator* pattern, which is a common way to adding behaviors to existing objects dynamically. For example, subsampling and oversampling, commonly used techniques to balance classes in learning classification problems, can be implemented by embedding a sequential Dataset Iterator and defining the method `next` such that `next` of the embedded is called a random number of times which depends on the value of `getOutput`. A big advantage of this dynamic implementation is that it fairly returns all the elements of the embedded Dataset Iterator during the iterations, a possibility forbidden by the commonly used static samplings. As shown in Figure 4.15, different Dataset Iterators are provided.

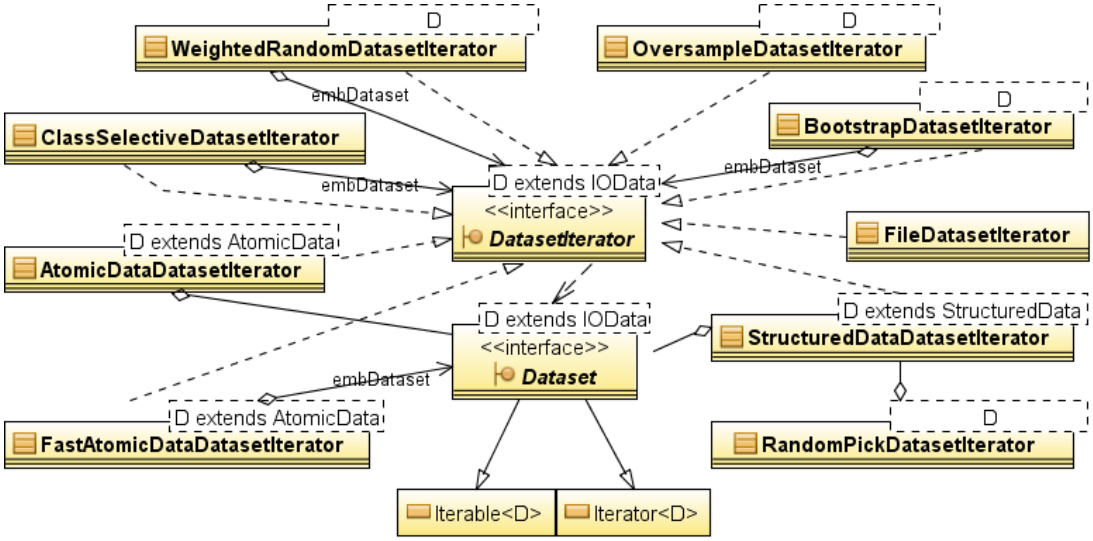


Figure 4.15: Hierarchy of standard Dataset Iterators in UML.

The main different types of dataset are `AtomicDataDatasetIterator`, which handles Atomic Data, whereas `StructuredDataDatasetIterator` handles Structured Data.

The implementation for Atomic Data is straightforward: iteration is completely delegated to the embedded Dataset, and encoding methods are dynamically applied to the current target to obtain the input and output windows. `FastAtomicDataDatasetIterator` is a faster static implementation which stores the input and output windows in an array instead of computing them dynamically at each iteration.

The implementation for Structured Data uses two level of iteration: the first on the embedded dataset, the second on the windows of the current Instance's Encoding. Windows are extracted sequentially in the base implementation, whereas a random number are "picked" up by each encoding by `RandomPickDatasetIterator`. Each Data provides only a subset of its inputs, according to a random choice performed in accordance with a specific parameter, say n. In particular, a random value k is extracted in the range $[0, n-1]$, then the inputs with index k, $k+n$, $k+2n$ and so on are provided to the learning algorithm. This implementation allows to efficiently simulate a complete random shuffling of the input values.

`FileDatasetIterator` directly reads data from a CSV file, and is used to directly test Learning Algorithms on pre-encoded datasets. Other implementations are general-

purpose decorators, and can be applied over any other Dataset Iterator.

`BootstrapDatasetIterator` is dynamically implements random bootstrap, which returns a specified random percentage (with duplicates), different at every iteration.

`WeightedRandomDatasetIterator` allows to associate a weight to each element of the embedded Dataset Iterator, which determines the probability of extraction of that element during an iteration.

`OversampleDatasetIterator` implements the oversampling technique described above, and finally `ClassSelectiveDatasetIterator` is a degenerate case of subsampling which iterates only on elements of a given class.

4.2.3.8 Gating Functions

Gating Functions define the domain of competence of an Expert. Gating Function modules are located in the package `game.gatingfunctions` and implement the interface `GatingFunction`, which exports the method `isIn`, which says whether a given Data is in the domain of competence of an Expert or not.

`EncodingBasedExpert` (and subclasses, including Learners) have a Gating Function; the method `predict` returns a `null` value if the input Data is not in the domain of competence. Combiners do not consider the null values, thus allowing to implement simple Mixture of Experts in which subsets of experts are specialized in different domains. A Learner filters the training set in order not to include the Data out of its domain of competence in the training.

The user could implement her own Gating Functions specialized on the type of the input Data considered. For the cases in which the gating feature is not desired, `TrueCondition` is a general-purpose Gating Function which includes all the Data in the domain.

`PythonLambdaCondition` is a flexible general-purpose solution which uses a Jython interpreter [1] to allow the definition of custom lambda function as inclusion condition for a given data. The lambda function is in the form:

```
lambda d: (boolean_expression)
```

[1] `http://www.jython.org`

in which the boolean expression can make use of all the public methods and attributes of the actual input data. For example, should the current data define an attribute `length`, the condition could be the application of a threshold of the length:

```
lambda d: d.length < 10
```

The drawback for the great flexibility of lambda conditions is a consistent loss of efficiency; for this reason, the definition of custom Gating Functions is suggested while working with big datasets.

4.2.4 Experiments

Experiment modules define the setup, training and test of a prediction technique. Every Experiment contains the configuration of a `Predictor`, which is a wrapper of an Expert which returns output in its natural format with the help of a Decoder, and a `SettingManager`, which defines general options such as logging, caching and the default Instance Data.

The interface `Experiment` defines the method `runSimulation`, which is launched when an Experiment is run. An Experiment is usually divided in three logic parts:

- *setup* – defines the general settings and the Predictor. It is usually carried out through the main GUI.
- *initialization* – initializes the Predictor. The predictor may be loaded from a file, and a training process may be started, if needed.
- *test* – the Predictor is tested, according to the Experiment policy and the type of Instance Data (further details are given in Section 4.2.5).

Experiments can be defined as modules located inside the package `game.controllers`. Figure 4.16 shows the available Experiment modules. A simple example of Experiment module is:

```
public class MyExperiment extends AbstractExperiment{

    public SettingsManager xmlsettings;
    public Predictor predictor;
```

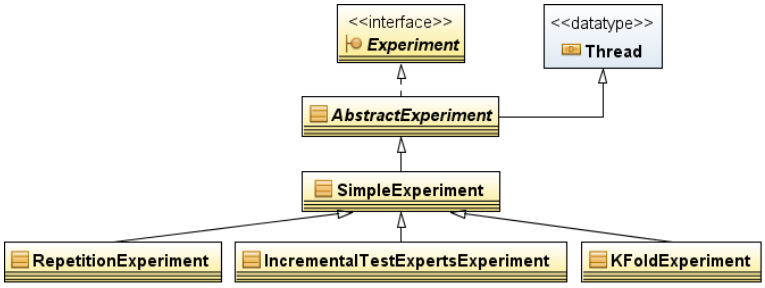


Figure 4.16: UML class diagram of the standard Experiment modules.

```
    ...

    @Override
    public void runSimulation() {
        //Global options initialization
        SettingsManager.setInstance(xmlsettings);

        //initialize predictor (training, if any, is made
        //automatically at the first prediction)
        Dataset testSet = ModuleManager.getInstance().getTestSet();
        predictor.predict(testSet.getCurrent());

        //test
        ModuleManager.getEvaluator().testDataset(predictor, testSet);
    }
}
```

The code above is basically the code of the module `SimpleExperiment`, excluding logging and exception handling. Other types of experiments may operate on the predictor or the test set, the current training set (which can be set on the Module Manager), or the Predictor.

`KFoldExperiment` defines a k-fold cross validation test by creating custom training and test set dividing the initial training set in subsets and initializing and testing different predictors with the same given configuration with them.

`IncrementalTestExpertsExperiment` iteratively builds and tests Predictors whose embedded Expert is a Combiner, adding an Expert to the Combiner at each iteration.

`RepetitionExperiment` repeats the same base Experiment a number of times, and displays the details of each experiment and the average of the result for all experiments.

4.2.5 Evaluation

Experiment modules evaluate predictors with Evaluators, which are modules able to test a Predictor on a single Instance Data or on a Dataset. Evaluators are located inside the package `game.evaluators`, and implement the interface `Evaluator`, which exports two methods:

- `getScore` – takes a Predictor and an Instance Data, and returns the score for that Instance as a floating point.
- `testDataset` – takes a Predictor and a Dataset, and returns the average result as a structured structure of type `Score`. A score implements some basic arithmetic operations with other objects of the same type, and can be printed out in a readable format.

Logging is an important operation carried out during the evaluation: the class `Msg` provides standard methods to display and log different types of result, in order to be retrieved and visualized once the experiment is terminated.

Three implementations of `Evaluator` are available in GAME 2.0:

- `ConfusionMatrixEvaluator` – implements evaluation based on confusion matrix (see Section 3.3.1. It can be applied to all `AtomicMulticlassData`, and its score displays the values of the confusion matrix together with standard measures of accuracy, precision, recall and Matthews Correlation Coefficient.
- `StructuredDataConfusionMatrixEvaluator` – implementation of confusion matrix for `StructuredMulticlassData`. Structured Data is considered as a sequence of `AtomicMulticlassData`.
- `ContinueOutputEvaluator` – can be applied to floating-point based Data (i.e. `DoubleData` and `VectorData`); measures the performance as Mean Absolute Error (MAE), Mean Square Error (MSE) and, for a Dataset, with the Pearson's correlation coefficient, considering the target and predicted output as two different signals.

4.3 GAME for Secondary Structure Prediction

In this section, the implementation of the problem of secondary structure prediction with GAME is detailed. Firstly, how the problem have been defined by its custom Data types, representing a protein with its primary and secondary structure, is described. Secondly, the alternative input, output and refinement encoding modules are presented. Thirdly, the standard prediction architecture and parameters, from which are built the prediction systems described in the following Chapters, is exemplified. Two case studies are finally presented: a reproduction of the predictor PSIpred and a comparison of combining policies.

4.3.1 Problem Definition

The module `HECProtein` describes the Instance Data type for SSP. As shown in Figure 4.17, Input Data is realized with the type `PrimaryStructure`, while Output Data is realized with the type `SecondaryStructure`.

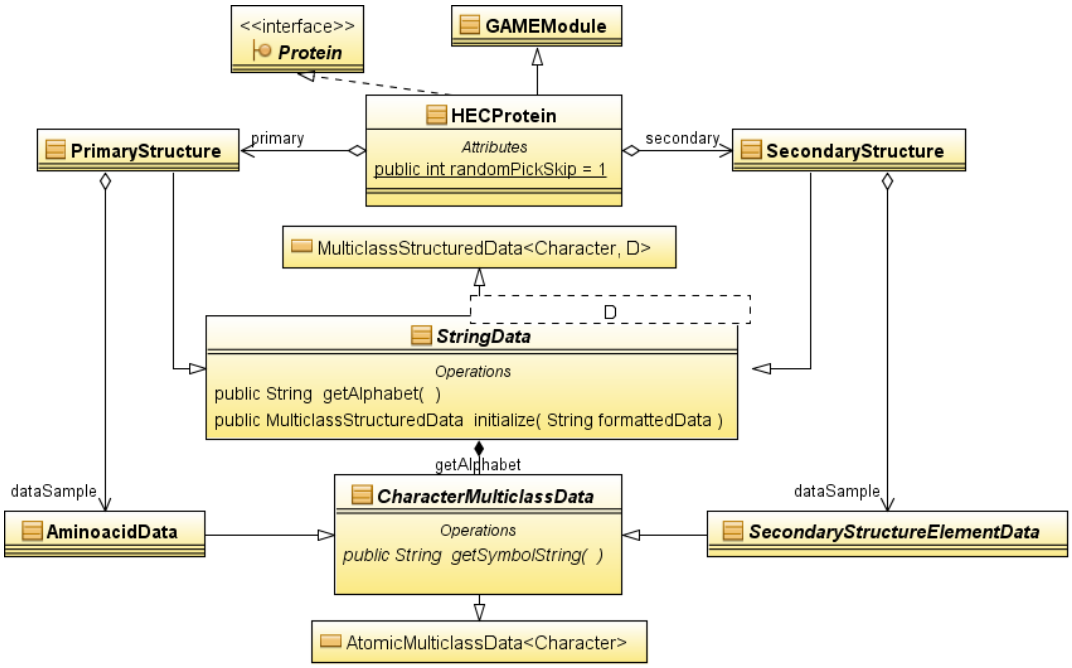


Figure 4.17: UML class diagram of the classes involved in the definition of the SSP problem.

Both primary and secondary structure are subclasses of `StringData`, parametrized

with atomic Data types derived from `CharacterMulticlassData`: `AminoacidData` and `SecondaryStructureElementData`. Subclasses of `CharacterMulticlassData` are required to define the alphabet of competence, which is *'ARNDCQEGHILKMFPSTWYV'* for amino acids and *'HEC'* for secondary structure elements.

The iteration policy of `HECProtein` is regulated by the parameter `randomPickSkip`. When the value is 1, a `StructuredDataDatasetIterator` is used. A `RandomPickDatasetIterator` is adopted otherwise, and `randomPickSkip` is passed to it as parameter. The random pick iteration in order to avoid possible skewness due to the related of consecutive encoding windows. A random pick value of 10 have shown to speed up the training of MLPs with backpropagation.

Although the `StructuredDataConfusionMatrixEvaluator` is perfectly suited, a custom Evaluator (`SSPTester`) is used to evaluate SSP, in order to adopt all the standard evaluation metrics (as described in Section 3.3.1).

4.3.2 Input Encoders

Input Encoders for SSP can be implemented as subclasses of `InputProfileEncoder`, parametrized for the `PrimaryStructure` Data type. Although general encoders such as `StructuredDataOneHotEncoder` and `CodeMatrixEncoder` are perfectly suitable, best performances are obtainable with custom techniques exploiting evolutionary information (see Section 3.4).

The following input encoding methods have been implemented so far:

- *PR* – `PrimaryStructureOneHotEncoder` is an alias of the standard `StreamOneHotEncoder` for primary structures. Each amino acid is associated to a bit in a vector with 20 elements.

- *SM* – `ScoreMatrixEncoder` encodes the k-th position of the target sequence with the k-th row of a substitution matrix (see Section 2.2.1).

- *APP* – `AminPropertiesProteinEncoder` encodes an amino acid with a four-value vector indicating the properties: hydrophobicity, volume, charge, number of atoms in the side chain (see Section 1.4.1).

- *CHE* – `ChemicalPropertiesEncoder` considers an amino acid as belonging to one of the eight chemical classes (see Section 1.4.1).

- *FR* – `FrequenciesEncoder` encodes the k-th position of the target sequence with the substitution frequencies (in the range $[0, 1]$) estimated in accordance with the k-th position of a corresponding multiple alignment.

- *(*OCC) – `OccurrencesEncoder` profile elements are calculated as substitution occurrences instead of frequencies. Basically, it is a Frequency Encoder with a different normalization criterion.

- *PSSM* – `PSSMEncoder` represents the standard position-specific scoring matrix (PSSM) computed by PSI-BLAST (see Section 2.2.4).

- *SB* – `SumBlosumEncoder` implements the encoding method proposed in MASSP3 (158), which linearly combines rows of a BLOSUM matrix, weighting them in accordance with the frequencies of the multiple alignment.

- *SLB* – similar to the SB, `SumLinearBlosumEncoder` linearly normalizes the scoring matrix before making the sum. See Chapter 5.

- *RSLB, WSLB, BPSLB* – `RankSumLinearBlosumEncoder`, `WeightedSumLinearBlosumEncoder` and `BandPassLinearBlosumEncoder` represent variants of SLB that use different threshold criteria on the e-value to select proteins to be included in the count of frequencies. These encoding methods are useful to study the impact of the inclusion of proteins at different evolutionary distances in the encoding.

The classes involved in the implementation of Input Encoders for SSP are shown in Figure 4.18.

Encoders based on multiple alignment derive from `PSIBlastEncoder`, which delegates the extraction of multiple alignments and PSSMs to a specific *Aligner* module. The proposed implementation `IASCPSIBLASTWebAppAligner` uses a web-app interface to PSI-BLAST (see 2.3.2.2) provided by an apposite web application. Further details about the implementation of the web application are given in Section 4.3.5.1.

4.3.3 Output Encoders

Output Encoders for SSP can be also be implemented as Profile Encoders. As for primary structures, general encoders such as `StreamOneHotEncoder` and

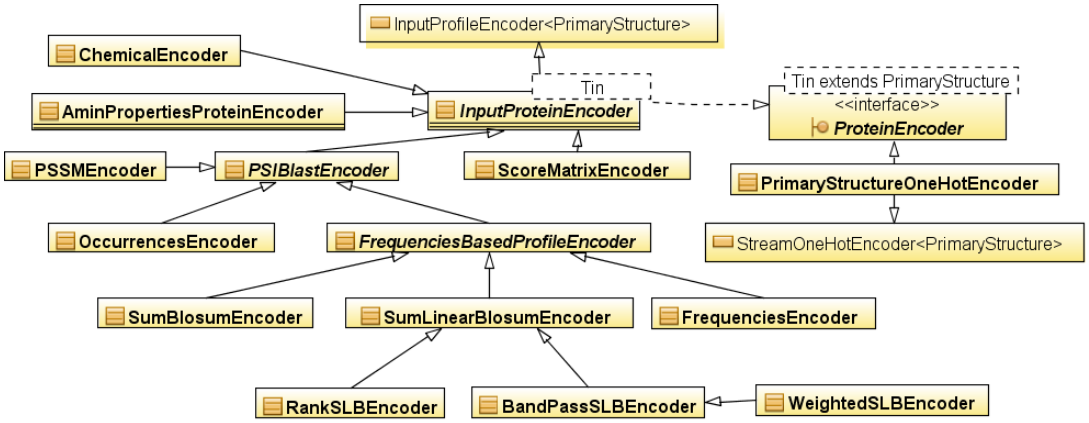


Figure 4.18: UML class diagram for the implementation of input Encoders for secondary structure prediction.

`CodeMatrixEncoder` can be used to encode secondary structures. These additional output Encoders have been implemented so far:

- *HEC* – `HECEncoder` is an alias of `StructuredDataOneHotEncoder` for secondary structures.
- *TR* – `HECTransitionEncoder` uses two additional bits to encode structural transitions. Implements the "Transition" coding described in Chapter 6.
- *BS* – `BetaSegmentsEncoder` is specifically devised to include the length and the position position of the beta-strand a beta-residue belongs to. Implements the "Beta Segments" coding described in Chapter 6.

Further details and experiments about the cited techniques are given in Section 6.1 and Appendix 4.4.1.

4.3.4 Refinement Encoders

The general-purpose modules `RefinementProfileEncoder` and `HybridRefinementProfileEncoder` can be directly used and adapted to refine SSP Experts.

Two custom alternative have been implemented:

- `PSIPREDS2SEncoder` is a `RefinementProfileEncoder` which implements global information as in PSIpred's secondary-to-secondary processing (see Section 3.5.2).

- `BIOInfoRefinementEncoder` is another `RefinementProfileEncoder` which adds biological information regarding amino-acids and proteins. The information to be added can be selected as parameters.

4.3.5 Standard Setup for a Secondary Structure Predictor

Given the leap in performance achieved by PSIpred, we assumed that any architecture based on the same principles should be capable of high performance. After implementing a PSIpred-like architecture and performing some preliminary experiments, the system has shown a performance comparable with state-of-the-art predictors and was therefore chosen as reference for all other experiments.

The P2S prediction is performed by a `LearningExpert` module[1], with MLP as prediction algorithm (module `BPNeuralNetwork` of GAME, with hidden layer set to 75 neurons). Input sliding window length was fixed at 15 positions. As for the output Encoder, one-hot coding with three values (*HEC*, GAME module `HECEncoder`) is normally used.

The S2S prediction is implemented with a `RefinementLearningExpert` module containing the P2S Expert. The hidden layer consists here of 55 neurons. A `PSIPREDS2SEncoder` module is assigned to the refinement encoding.

Combination, where applied and unless otherwise stated, is made with an AVG combination policy (module `AveragingCombinerExpert`).

4.3.5.1 Encoding Implementation Details

Experiments have been performed with the following encoding methods: SLB, PSSM, and FR[2].

Frequency profiles and PSSM have been obtained with the IASC-BLAST, a web interface for PSI-BLAST[3]. IASC-BLAST permits to select the inclusion threshold, the number of iterations and the dataset in which to perform the search. The standard values are used for the rest of the parameters. The application, written in Python[4], runs a stand-alone executable of PSI-BLAST *blastpgp* version 2.2, as downloaded from the

[1] The `LearnerExpert` is used instead where a meta-learning technique is applied.

[2] See Section 4.3.2 for the details about the implementation of encoding techniques within the framework.

[3] `http://iasc.diee.unica.it/blast/`.

[4] `http://www.python.org`

NCBI ftp server. Frequency profiles are extracted by parsing the file resulting from the -Q switch, while the PSSM is obtained passing the parameter -C and using the utility *makemat*. The result is given in the standard BLASTXML format, modified in order to include PSSM and frequency profiles. The *PDB*, *uniref50* and *uniref90* (see Section 2.1) datasets are available. *uniref50* and *uniref90* filtered versions, say *uniref50filt* and *uniref90filt*, are also available. Filtering has been carried out as in PSIpred with Jones' program PFILT (159) to remove trans-membrane proteins and simple sequence regions.

SLB and FR make use of the frequencies provided by PSI-BLAST. SLB is implemented in accordance with the pseudo-code reported in Figure 1, while the FR encoding is obtained by simply normalizing the frequency given by PSI-BLAST in the range $[0, 1]$[1].

The PSSM encoding algorithm has been implemented in accordance with the PSIpred paper and reverse-engineering the relevant code found in a recent implementation.

4.3.5.2 Training Technique and Parameter Setting

In the MLPs used in the P2S and S2S modules, learning rate is set to 0.001 (initial value) and momentum to 0.1. The learning rate is adjusted between iterations with an inverse-proportion law.

With the goal of improving the ability of the training algorithm to escape from local minima, at each iteration the training set is randomly "shuffled", so that the learning algorithm is fed with the same set of proteins given in a different order. Furthermore, a "Random pick" iteration criterion ($n = 10$, see Section 4.2.3.7) is adopted in order to efficiently randomize the training within single proteins.

To prevent the training process from stopping with a local oscillation of accuracy (evaluated on a validation set consisting of 10% of the training proteins), the following strategy has been adopted: weights are recorded when a minimum on the validation set is encountered, but the training continues until the error on the validation set exceeds a dynamic threshold that decreases as the iterations proceed. A separate test set is used for all final assessment of results.

[1] In the event that no alignment is found at position i of the target protein (namely, $freq[i, \_]$ is set to all zeroes), $freq[i, j]$ is set to 1, j being the index of the amino acid that actually occurs at position i in the target protein.

4.3.6 Case Studies

4.3.6.1 Implementing PSIpred with GAME

All choices made to design and implement PSIpred can be reproduced by customizing the GAME generic architecture. Figure 4.19 illustrates a feasible realization of it. In particular, PSIpred can be reproduced by customizing a Refiner (the structure-to-structure expert), which in turn encapsulates a Combiner made up of four experts, each one entrusted with performing a sequence-to-structure prediction (input: PSSM, output: HEC). The combiner adopts the *Averaging* output-combination policy. A

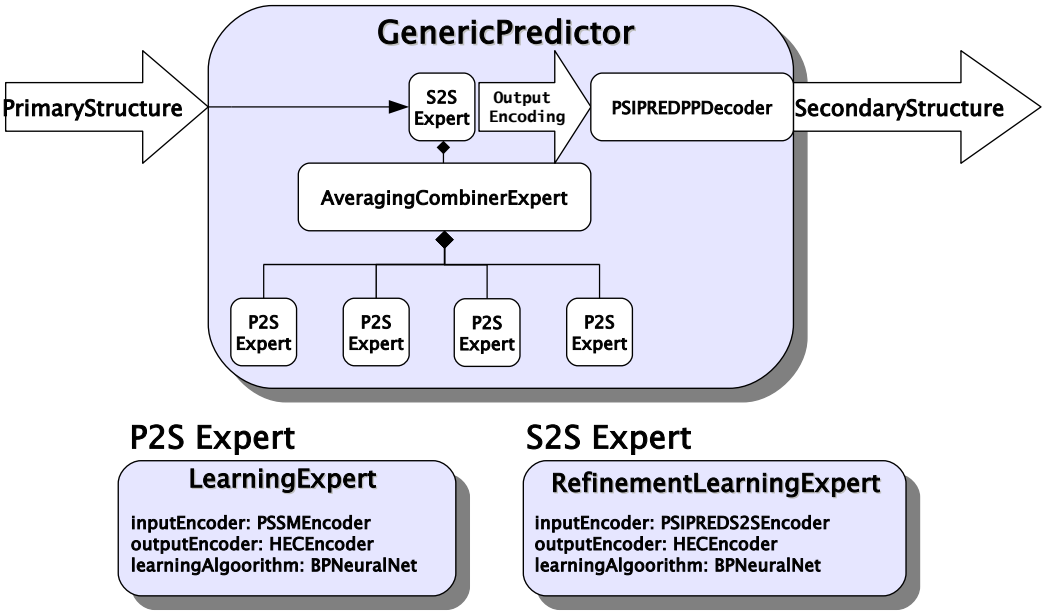


Figure 4.19: PSIPRED with GAME

custom decoding module, `PSIPREDPPDecoder`, must be implemented in order to apply the custom post-processing performed by PSIpred on the prediction.

4.3.6.2 Combiners comparative study

Here two comparative examples for the problem of secondary structure prediction carried out with GAME are presented. Results for different combining strategies are compared, maintaining the rest of the system unchanged. Merging policies are: majority voting (MAJ), averaging (AVG), weighted averaging (WAVG), elite averaging with

25% of predictions (EA25), elite averaging with 50% of predictions (EA50), elite majority with 25% of predictions (EM25), elite majority with 50% of predictions (EM50), and stacking by means of an MLP trained on a dataset of 2000 proteins (NEU).

The first test compares the different combination policies for 14 homogeneous experts. Each Expert, realized with a `LearningExpert` with PSSM encoding and MLP as Learning Algorithm, is trained with a different set of 250 proteins with a similarity not greater than 25%. Experts have been first trained within the same AVG Combiner using the meta-learning technique `DatasetSubsetMetaLearner`, which have been serialized in XML format and then reloaded for the other experiments. All Experiments automatically save the XML serialized Predictor and inner Expert after initialization is concluded. The Expert can be loaded from the graphical interface at any point of the Expert tree, and it can be further configured within the GUI. In the case of combiners, Experts can be added and deleted freely before running another experiment.

Figures 4.20 and 4.20 show the configuration steps needed to reproduce this experiment using the GAME GUI.

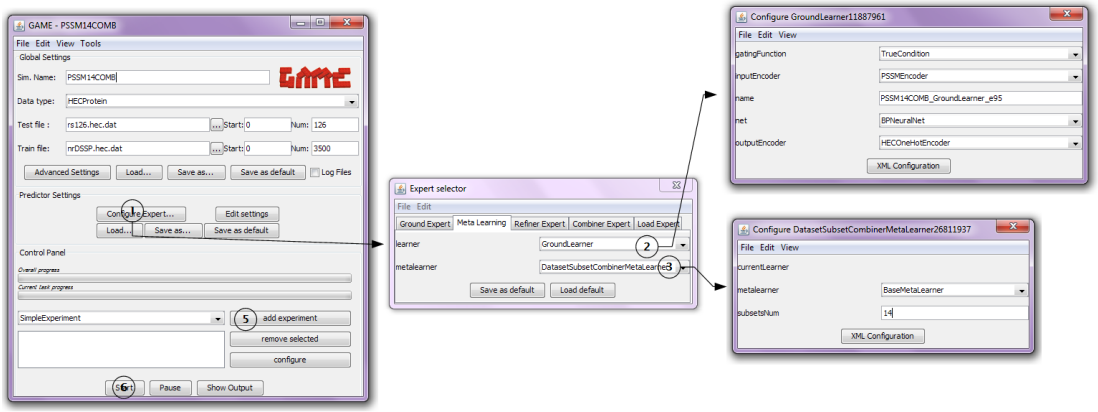


Figure 4.20: Setting up an experiment with 14 experts trained with different dataset subsets.

Figure 4.22 shows the result for the homogeneous comparison. It is noticeable a general equivalence of the different techniques, with a slight prevalence of stacking.

Figure 4.23 reports a similar experiment, performed with a set of experts in which the selected input encoding method varies. This recalls the input encoding variation studied in (114). In particular, different sets of 14 experts have been trained using standard PSSM, FR and SLB; then, all these experts have been combined. Also here,

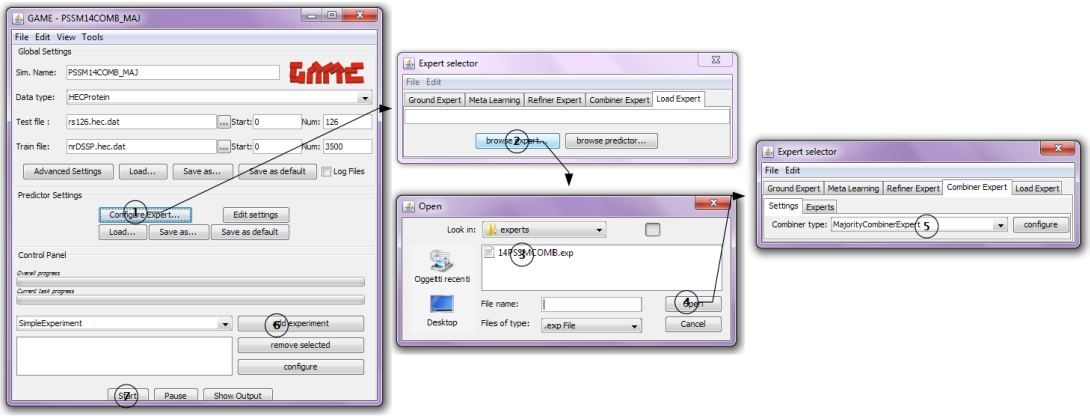


Figure 4.21: Loaded and reconfiguring a previously trained Combiner in order to change the combination policy.

the different combining techniques show quite similar result for all the relevant metrics. It can be noticed an average improvement of the performances in the latter experiment, which may be related to the heterogeneity injected from the use of different encoding techniques, or more simply to the increased number of Experts combined.

4.4 Perspectives

"Real world" applications give birth to challenging problems that can take advantage of machine learning techniques. Different widely acknowledged tools make the process of training and evaluating a ML technique, and comparing it to other techniques, really fast and easy. Unfortunately, the well-known existing tools for ML do not help in the feature extraction and labeling tasks, which can be critical especially when dealing with structured data typical of RWPs, such as texts, signals, images, or sequences. With the desire of realizing a system without this limitation, revealed during the first attempts at building secondary structure prediction systems, I have developed a new software architecture and framework, called GAME. GAME permits to easily implement and compare different modules performing the same task, and provides a graphical environment that assist the user setting up experiments involving custom configuration and connection of different automated experts (i.e., classifiers or predictors).

The funding principles of the architecture have been presented, and their implementation in the framework has been described. Then, the standard modules available

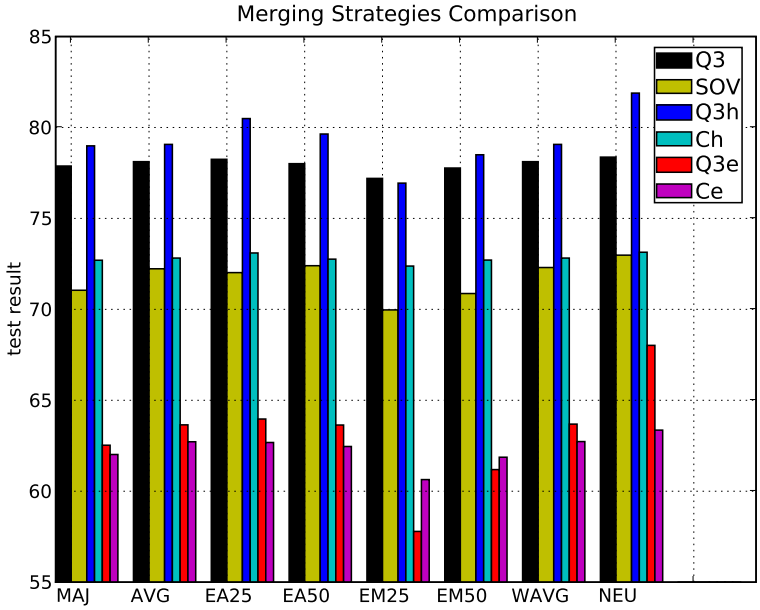


Figure 4.22: Prediction results obtained by using different combination strategies.

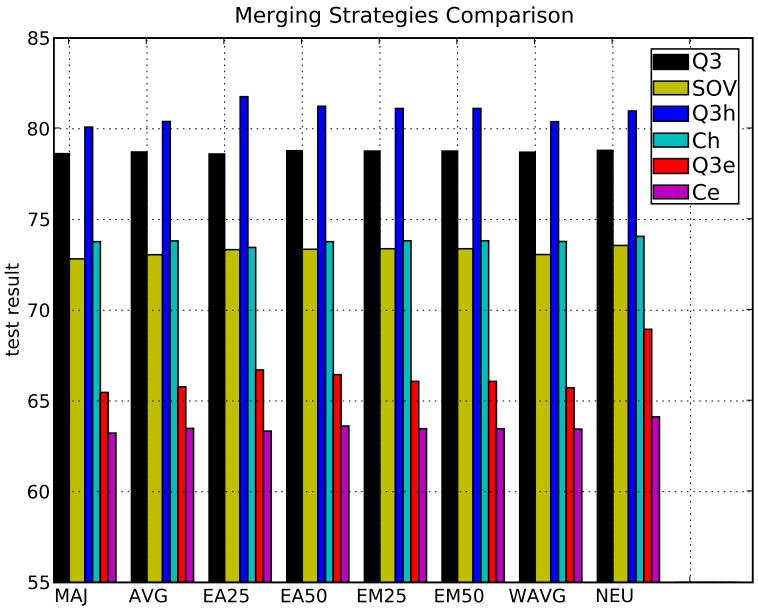


Figure 4.23: Prediction results obtained by using different merging strategies (with different encoding methods)

have been described, explaining how they can be composed and extended to build an actual prediction system. Finally, the application of the framework to the problem of secondary structure prediction has been detailed, presenting two different case studies as examples.

GAME is now at its second release. With this release, after three years of development and adaptations, GAME can now be easily used for a wide variety of problems belonging to various application fields. At present, GAME has been applied to different prediction problems, mostly belonging to the field of bioinformatics. Other than secondary structure predictors, have been realized with GAME: (i) an antibody packing angle predictor (ii) a simple optical character recognition system[1] (iii) a beta-contact predictor (described in Appendix A) (iv) a predictor of disease-causing single amino acid polymorphisms.

The weaker points are related some accessory mechanisms, such as logging and result presentation, that could be better integrated with the rest of the system. As for stability of the software, very few issues still arise sometimes, mainly related to old modules that have missed to follow some of the numerous re-engineering steps.

The main future objective is the realization of new problems and new modules, in order to make the use of the tool more appealing. With this in mind, GAME has been released with open source license[2], with the hope that it will be adopted by the community. GAME home page is `http://iasc.diee.unica.it/GAME/`.

[1] Realized by Alberto Malerba.

[2] `http://game2.sourceforge.net/`

4.4.1 Experimenting Error-Correcting Output Codings for Eight-Class SSP

Error-correcting output codes decomposition (160) is a general technique which permits to build ensembles for multiclass classification from binary classifiers. According to the information theory, properly adding extra information to a coded message permits to obtain codewords with error-correcting capabilities. In particular, the original message can be reconstructed in presence of a number of errors at the receiver lower than the minimum hamming distance for that code.

The idea of ECOC decomposition is to exploit the error-correcting capability in classification problems, by using redundant codes to represent the different classes. For a problem with ω classes, up to $2^{\omega-1} - 1$ elements can be used to compose a non-synonym codewords for that classes. The resulting codes can then be used to build ensembles of binary classifiers, each one predicting an element of the binary word.

As no proper ECOC can be devised for 3-classes problem, a predictor for the 8-classes problem (according to the DSSP set of annotations) has been devised and experimented with ECOC decomposition. The problem with eight different classes is much less studied than the three-class version, and very poor results have been obtained (about 63% Q_8 was claimed by a specific version of SSPRO).

The dense-random strategy (142) has been used to obtain a coding matrix with code words of 30 elements[1]:

```
H: 11101101000010101001001111 0011
G: 01000001011111110000011 0010100
I: 01111000101110101011010 0010111
E: 10000100000101100001010 0100000
B: 01000100111000001110111 1110111
T: 00111001100001111110010 1110001
S: 01011111110110100101110 0100100
C: 10110110111001111110001 0100100
```

This matrix was used to build a predictor based on an ensemble of 30 experts. The predictor was implemented with the GAME framework, applying the ECOC decomposition meta-learning technique (`ECOCDecompositionMetalearner`) to a P2S module

[1] A special thanks to Nima, who provided the matrix to use in the experiments.

configured in accordance with the standard setup given in Section 4.3.5, with output encoding being delegated to the `CodeMatrixProfileEncoder` module. The ECOC predictor has been compared with a single S2S Expert (*SINGLE*) and with a PHD-like system combining 10 S2S Experts (*AVG10*). Table 4.1 shows the result for a 3-fold cross validation experiment in a training set of 3500 unrelated proteins. Unfortunately,

Method	$Q_8(\mu)$	$Q_8(\sigma)$
SINGLE	63.900	0.800
AVG10	64.667	0.850
ECOC	64.633	0.971

Table 4.1: 3-fold cross validation results for a dataset of 3500 unrelated proteins.

although cutting-edge, the performance for the ECOC predictor did not show any increment respect to the standard PHD-like system. This is not enough to assess the success for the ECOC technique, since it requires much more computational resources than its acknowledged counterpart. This missed increment may be related to the lack of the refinement step, which is known to be able to boost prediction performance.

Further work may be related to the use of a refinement step together with an ECOC architecture, and to the experimentation of alternative error-correcting codings.

Chapter 5

Exploiting Inter-Sequence Information in Secondary Structure Prediction

Input representation (also called pre-processing or input encoding) is essential to represent the sequence-to-structure and inter-sequence information contained in the primary structure in a form directly usable by the underlying predictor.

A protein sequence can be represented by a *profile*[1], which may encode amino acids in two ways: (i) position independent, which means that the same amino acid is always represented in the same way regardless of its position, and (ii) position specific, which typically exploits the evolutionary information contained in multiple alignments.

The capability of exploiting evolutionary information was the main innovation introduced in the 1990s for representing proteins. Homologous proteins share a common three-dimensional structure, and this is reflected in the similarity between primary sequences (inter-sequence correlation). As a result, given a protein sequence, a multiple alignment obtained from a set of similar sequences found in a sequence database is expected to contain the information about the substitutions that occurred within the protein family during its evolution without compromising the structure.

To predict the secondary structure, the substitution frequencies extracted from a multiple alignment considered representative of the target sequence in hand have

[1]The term 'profile' is generally used in the literature as a synonym of PSSM, but is intended here in the more general sense of a matrix which associates a vector to each element of the protein chain.

proven to be effective inputs. The i-th position of a target protein being encoded with a real vector of 20 elements (each one approximating the probability that a given amino acid occurs at position i in the family to which the target protein belongs), a protein of length M can be represented by an array of M x 20 real-values. Substitution frequencies are a notable example of position-specific encoding, as the same amino acid may be represented in different ways, depending on its position within the protein being encoded.

Analyzing the history of secondary structure predictors, most of the reported improvements are related to input representation. The use of evolutionary information in the task of SSP dates back to 1976 (161) and was applied in 1987 by Zvelebil et al. (162). PHD (109), the first system able to break the barrier of 70% Q_3, used multiple alignment frequencies obtained from the HSSP databank (138) to encode target proteins.

After PHD, frequency-based encoding based on multiple alignment has become the standard de-facto of any input representation for SSP. A further improvement in PHD performance, specifically, from 70% to 72% Q_3 (163), was obtained by adopting BLAST (Section 2.3.2.2) as the search algorithm and CLUSTALW (28) as the alignment algorithm. PSIpred (113), inspired by PHD, has been able to reach an accuracy of 76% by directly using Position-Specific Scoring Matrices (PSSM, see Section 2.2.4) built by PSI-BLAST, the iterative version of BLAST. The improvement obtained by PSIpred depends also on the fact that the database containing proteins has been filtered by removing the potential causes of the drift: redundant sequences, simple sequences, and transmembrane regions.

Since PSIpred, PSSM has been acknowledged to be the most effective encoding method available for SSP and has also been used by other software architectures and systems (e.g., SSPRO (115), (123)).

Predictors that exploit different encoding methods and formats have also been devised. For instance, Karplus et al. (35) used an encoding built upon Hidden Markov Models (HMMs) (164), whereas Cuff and Barton (114) assessed the impact of combining different encoding methods in the task of secondary structure prediction. We contributed to the encoding topic by proposing Sum-Linear Blosum (SLB) (3).

An up-to-date position-specific input representation task typically involves the following activities: *search*, *alignment*, and *encoding*.

5.1 Encoding and Slice Decomposition

A target sequence is typically turned into fixed-length vectors to make it easier for the adoption of well-known learning techniques to operate on fixed-length inputs. As sketched in Figure 5.1, sliding windows are the most common technique for obtaining fixed-length real-valued vectors (slices hereinafter) from input profiles. It is worth noting that, while generating slices for the first amino acids, part of the sliding window actually points outside the input profile. For a window with $length = 7$, the first, second, and third amino acids require a specific treatment. The same problem occurs while encoding the amino acids at position $M - 1$, $M - 2$, and $M - 3$. A typical solution to this problem consists of adding an additional input entrusted with making the border-line conditions explicit.

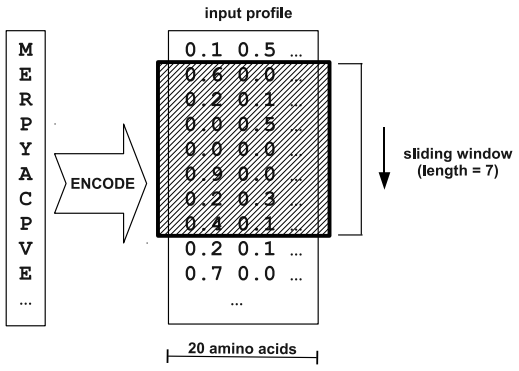


Figure 5.1: Using sliding windows to extract fixed-length slices from input profiles: a window centered on the fifth amino acid of the target sequence.

5.2 Encoding Multiple Alignments

Profiles are a way of representing multiple sequence alignment information, a profile being represented by a matrix with M rows (one for each position of the target sequence) and 20 columns (one for each amino acid). Profiles typically may encode (i) position-independent information, (ii) position-specific information, or (iii) both. Profiles that embed position-specific information are also called Position-Specific Scoring Matrices (PSSMs).

"One-hot" is a relevant position independent representation typically used to encode data which represent multi-class values. A vector of 20 bits is required to encode a specific residue occurring in a protein; thus, a protein of length M can be represented by a profile of M x 20 bits. Another position-independent representations of amino-acids may consider their chemical or functional properties; a one-hot coding may be applied after a clustering of amino acids according to their chemical or functional properties. Physical values such as mass, volume, number of atoms in the side-chain, hydrophobicity, charge, pK values can all be used to encode a residue. General substitution matrices rows, intending the row j as the propensity for the amino acid j to be substituted by other amino acids, can be also used as position-independent encodings.

A typical example of a profile that embeds only position-specific information is the frequency-based encoding (FR hereinafter) used in PHD. Here the relevant biological information comes from a multiple alignment of the target protein with evolutionary related proteins found after a similarity search in a sequence database. Assuming that a multiple alignment ranges over M positions (from 0 to $M-1$) and contains L sequences (from 0 to $L-1$), the encoding process can be thought of as a function that maps the alignment to a profile:

$$encode : Alignment[M, L] \rightarrow Profile[M, 20] \tag{5.1}$$

In the absence or deficiency of position-specific information (i.e. when the search did not find a sufficient number of significant hits), FR encoding may not work as expected. In this case, improvements can be obtained by integrating both kinds of information. The PSI-BLAST PSSM is the most widely acknowledged example in this category. As well as providing a multiple alignment at the end of a similarity search, PSI-BLAST may optionally output the characteristics of the family to which the target protein belongs in the form of a PSSM. Any such PSSM contains both position-independent and position-specific information, the former taken from "pseudo-counts" extracted from BLOSUM substitution matrices, and the latter taken from frequency counts. Here, the relevant biological information is contained in the SM adopted for encoding the amino acids that occur in the target protein and in the multiple alignment. Hence, in this case, the encoding process can be thought of as a function that maps the alignment and the scoring matrix to a profile:

$$encode : SM[20, 20] \times Alignment[M, L] \rightarrow Profile[M, 20] \tag{5.2}$$

Depending on the quality of the result (and on relevant parameters of the algorithm), the final PSSM given by PSI-BLAST may actually contain different "quantities" of the two kinds of information: the fewer significant hits found in the search, the greater the amount of position-independent information embedded into the PSSM. The PSI-BLAST PSSM encoding is widely acknowledged as one of the best encoding method. Unless otherwise stated, "PSSM" will refer to PSI-BLAST PSSMs hereinafter.

After agreeing that any modern encoding method should integrate different kinds of information, our claim is that more effective encoding methods might be devised for particular predictors. This claim is supported by the fact that PSSM was originally optimized for an iterative search procedure, which gives priority to speed rather than precision. Its use in encoding tasks is a kind of side effect, although one at which it is remarkably effective.

5.3 SLB: A Novel Encoding Method

The starting point of our work was the SUM-BLOSUM procedure used by MASSP3[1] (125), called SB hereinafter. To summarize this method, which integrates position-independent with position-specific information, recall that the encoding is performed row-by-row over the matrix, A, that represents the alignment.

We will represent as $A[i, \_]$ the i-th row of the multiple alignment, which contains the elements: $A[i, j], j = 0, 1, ..., L-1$ (where L is the number of sequences). Hence, given $A[i, \_]$, for each $j = 0, 1, ..., L-1$, the row of the BLOSUM80 matrix that corresponds to the amino acid found at position $A[i, j]$ is added to the current profile. The overall profile is obtained by repeating this operation for $i = 0, 1, ..., M-1$ (where M is the length of the alignment) and normalizing by the number of proteins (L) that occur in the alignment.

As SB did not show any advantage over the standard PSSM, we have focused on the task of improving the SB encoding –with the goal of embedding it in the next release of MASSP3.

SLB, the proposed encoding method, is a straightforward variant of SB. The main differences between SLB and SB are: (i) the substitution matrix is not in log-odds

[1] MASSP3 stands for Multi-agent Secondary Structure Predictor with Post Processing.

format, such that summation occurs in the domain of odds rather than log-odds[1], (ii) SLB uses frequencies rather than the underlying multiple alignment while repeatedly summing up rows of a substitution matrix, and (iii) the normalization is performed using a logistic sigmoid function.

As shown in Listing 1, the SLB encoding function takes as input the target sequence and the multiple alignment frequencies (*freq*) computed by PSI-BLAST, normalized in $[0, 1]$. Its output is the corresponding profile (*profile*). To perform the encoding, first

```
def SLB(seq, freq):
  """
  Sum-Linear-BLOSUM Encoding Procedure:
         IN = target sequence x frequencies,
         OUT = profile
  """
  profile = Profile(rows=len(seq), cols=20, fillWith=0)
  substMatrix = BLOSUM(load='blosum62', format='raw')
  substMatrix.adjust() #use one-hot encoding for columns with gaps only
  substMatrix.pow(2)  #use squared values
  for i in range(len(seq)):
    for j in len(AMINOACIDS):         #AMINOACIDS = 'AR...VY'
      profile[i][j] = profile[i][j] + freq[i] * substMatrix[j]
  profile.normalize() #normalize the profile with a sigmoid
  return profile
```

Listing 1: The pseudo code concerning the proposed encoding method (in a Python-like syntax).

the BLOSUM62 substitution matrix is loaded and converted from log-odds to odds, then an empty profile is created. The main iteration proceeds along the protein by progressively setting up each row of the profile. In particular, given the i-th row, the inner iteration sums up the j-th row of the BLOSUM62 matrix (j=0, 1, .., 19), weighted with $freq[i, j]$ –namely the frequency of the j-th amino acid measured at the

[1] A real-valued BLOSUM62 representation (i.e. without the limit due to the 4-bit representation adopted in the PSSM and SB encodings), namely *blosum.sij*, is available from the BLOCKS database site http://blocks.fhcrc.org/.

i-th position. Finally, the profile is normalized with the following function:

$$f(x) = sigm(-x) = \frac{1}{1 + e^x} \tag{5.3}$$

5.4 Experimental Results

Experiments have been realized with the GAME framework (see Chapter 4 and in particular Section 4.3 for modules specifically related to secondary structure prediction).

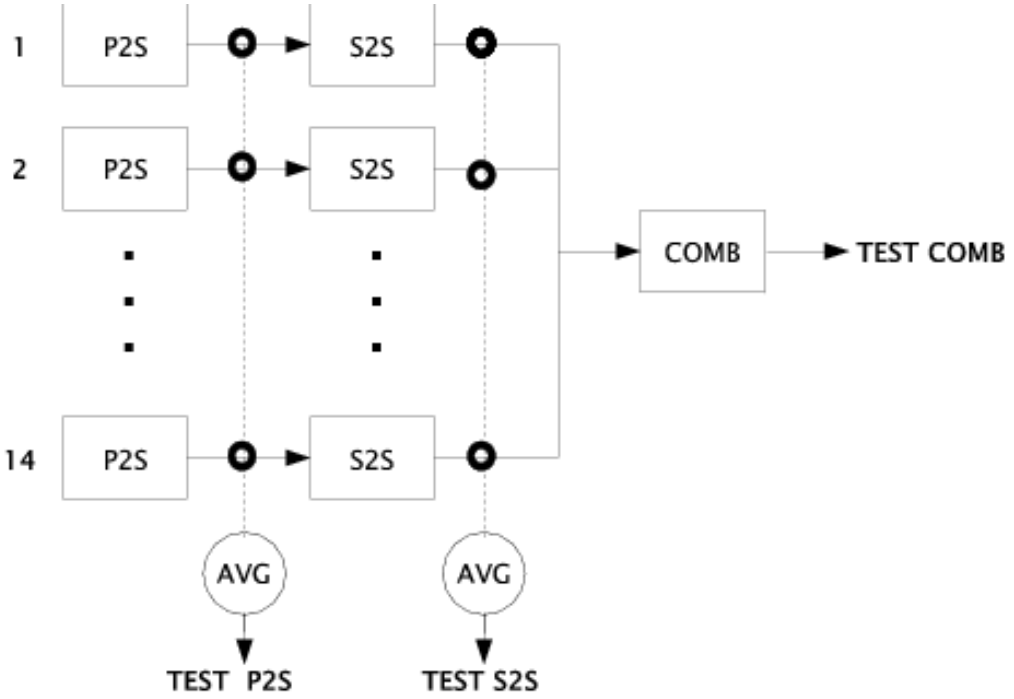


Figure 5.2: The GAMESSP benchmarking architecture.

As shown in Figure 5.2, GAMESSP embodies (and combines the output of) several prediction units. Each unit performs two kinds of transformations, which occur in a pipeline: primary-to-secondary (P2S) and secondary-to-secondary (S2S) structure prediction. The P2S and S2S modules have been implemented according to the standard setup shown in Section 4.3.5. The selected data repository was PSI-BLAST searches is *uniref50filt* with the inclusion threshold set to 10^{-5} and the number of iterations set to 3. In the system used for benchmarking, 14 of such units are used in parallel and results are combined.

The main output of GAMESSP is denoted in Figure 5.2 as "Test Comb". For benchmarking purposes, the system has been equipped with additional outputs, namely "Test P2S" and "Test S2S", which have been provided to give further information about the statistical significance of the experimental results. In particular, the former gives information about the (average) behavior of the P2S modules, whereas the latter gives information about the (average) behavior of the S2S modules.

The current release of the GAMESSP predictor is freely usable through the online web interface at `http://iasc2.diee.unica.it/ssp`.

5.4.1 Benchmarking Issues

To assess the effectiveness of our encoding method we performed experiments against other relevant methods. In order to guarantee the statistical significance of experimental results, we adhered to the following policy: (i) the same boundary conditions must be guaranteed for any encoding method, ensuring that no advantage is given to a particular method, (ii) an encoding method must be tested in different operational contexts, (iii) experimental results must show statistical significance.

As for the first issue, the same alignments and frequencies have been used for feeding all selected methods. The second issue has been dealt with by performing several experiments with different sets of training proteins of different sizes. Statistical variance has been controlled by using 14 prediction units, separately trained.

5.4.2 Ensuring Statistical Significance of Experimental Results

As previously stated, to ensure statistical significance of the experimental results, 14 prediction units have been independently trained on different subsets of randomly-selected inputs.

The dataset used for benchmarking purposes was extracted from the nrDSSP dataset (127), consisting of proteins with similarity $\leq 25\%$ and length ≥ 80 residues. The whole dataset, including 3925 proteins (available in the DSSP database during September 2004), was randomly shuffled and then training, validation, and test sets were extracted from it. In particular, 3500 proteins were used to build 14 distinct subsets of 250 proteins (numbered from 1 to 14).

Using a variable number of training proteins, say $N = 25, 50, 75, ..., 250$, different test runs have been performed by repeatedly selecting training and validation sets (for each prediction unit $e_i, i = 1, 2, ..., 14$) according to the following rule:

- *i-th training set.* The first N proteins are extracted from the i-th subset,
- *i-th validation set.* The $(1 + i \ mod \ 14)th$ set is used as the i-th validation set (consisting, in any case, of 250 proteins).

The rest of the available proteins (i.e. 425 proteins) has been used as a test set in all experiments.

5.4.3 Experimental Results

The impact of encoding has been assessed for both the P2S (Figure 5.3) and the S2S (Figure 5.4) transformations. For the sake of brevity, only the Q_3 performance index is reported. To see the potential of SLB in a real context, the impact of encoding on the

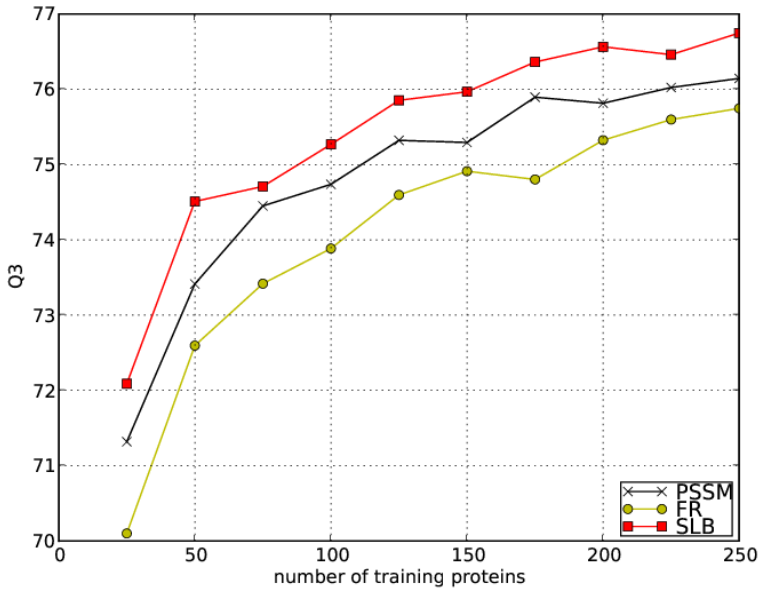


Figure 5.3: Average performance of P2S prediction units, given in terms of Q_3, obtained by using different encoding methods.

performance of the overall secondary structure predictor is also assessed (see Figure 5.5

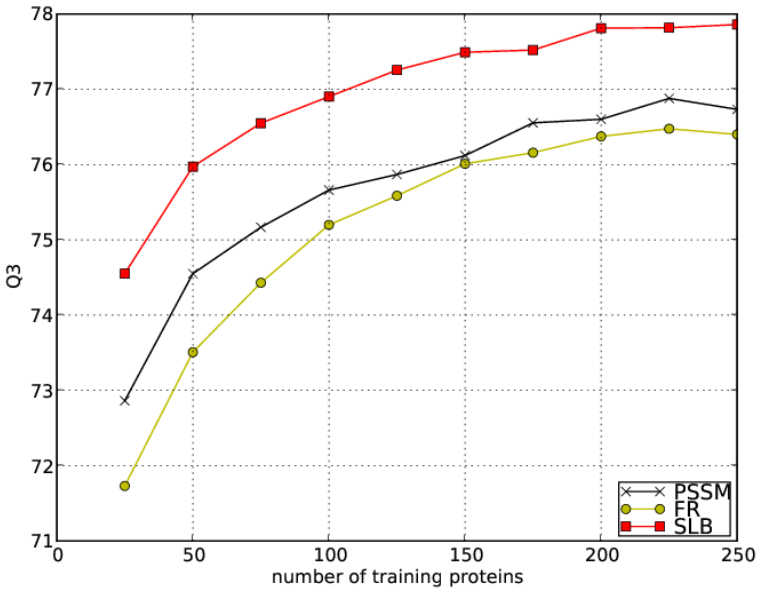


Figure 5.4: Average performance of prediction units, given in terms of Q_3, obtained by using different encoding methods.

and Figure 5.6 for Q_3 and SOV, respectively). Furthermore, Figure 5.7 reports the average performance of the backpropagation algorithm on the validation set.

Table 5.1: Some results obtained by running the GAMESSP online predictor on relevant data sets.

Test Set	Q_3	SOV	Q_{3_h}	Q_{3_e}	Q_{3_c}	C_h	C_e	C_c
RS126	80.70	76.17	83.40	72.56	82.98	0.770	0.677	0.631
CB513	81.31	78.69	88.27	73.16	80.62	0.771	0.688	0.641
ALL EVA (2217 prots.)	80.69	78.39	84.28	71.78	82.32	0.762	0.689	0.629

Table 5.1 reports the performance of the the GAMESSP online predictor on some well-known test sets[1]. Although compliant with the architecture illustrated in Figure 5.2, the online predictor actually embodies 7 prediction units –obtained by running a 7-fold cross-validation on the reference data set (i.e. nrDSSP). In particular, at the end of each cross-validation step, the best configuration found by the training algorithm

[1]Although these sets were not used to train the predictor, the separation between the training set and these set was not assessed. Average performance on new proteins is expected to be lower.

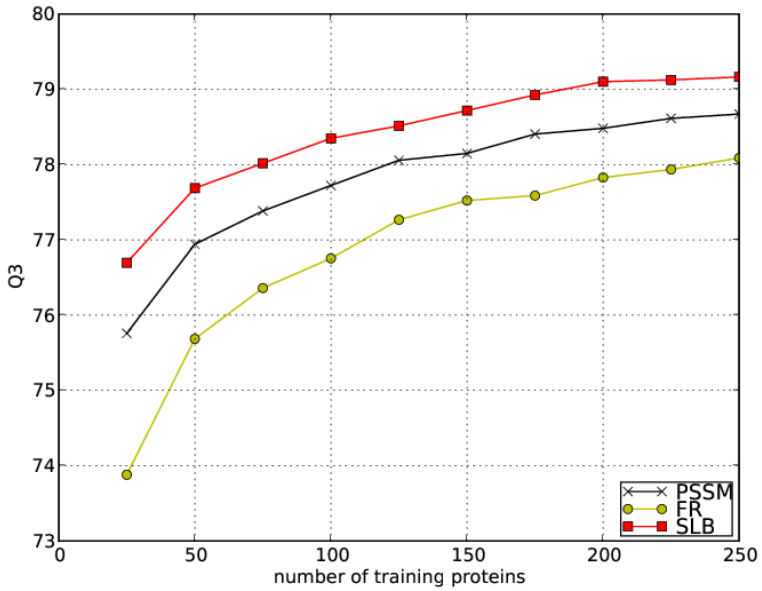


Figure 5.5: Overall system performance, given in terms of Q_3, obtained by using different encoding methods.

has been recorded. In so doing, together with the average result, the cross-validation process actually yielded the configuration of the above prediction units.

5.4.4 Discussion

Experiments show that the improvement is more evident with a limited number of training proteins: the smaller the number of proteins the greater the difference between SLB vs. PSSM or FR. In particular, the outputs drawn from P2S and S2S (see Figure 5.3 and Figure 5.4) show a difference of about 1.5% between SLB and its best competitor, i.e. PSSM. The difference between SLB and other encoding methods is less evident when the output is taken after the combination stage. This can be explained thinking that the positive effects of unit combination tend to hide the benefit of encoding. Nevertheless, even after combination, there is a small improvement (about 0.5% over PSSM in the worst case) between the performance obtained by adopting SLB vs. other encoding methods.

On the basis of experimental results, our current hypothesis about the superiority of SLB lies in the conjecture that it is able to perform a better generalization, which is

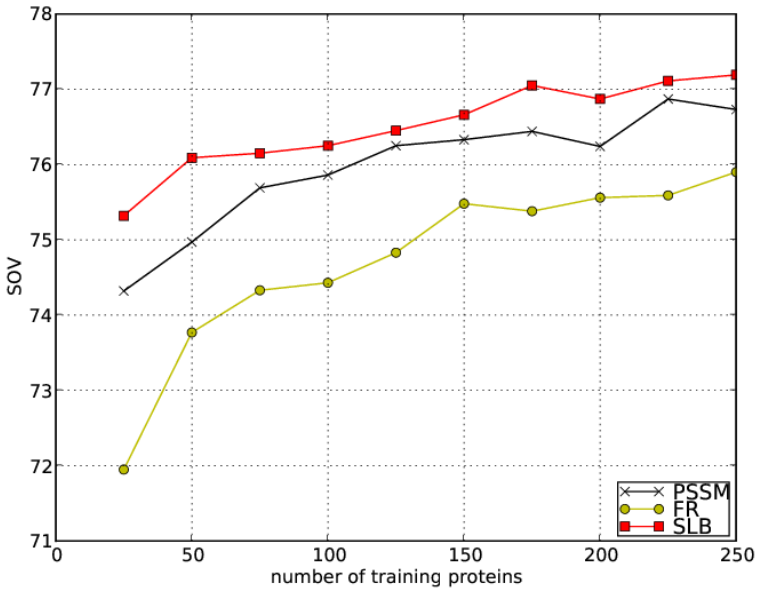


Figure 5.6: Overall system performance, given in terms of *SOV*, obtained by using different encoding methods.

more evident when the number of training proteins decreases. Further, to investigate the conjecture about the generalization ability, we reported in Figure 5.7 the trend of the Mean-Square-Error for both training and validation set for SLB and PSSM over 100 steps (evaluated over a training set of 250 proteins and on a validation set of 250 proteins). Training and validation sets do not share proteins with similarity $> 25\%$. The graph suggests that SLB avoids overfitting, which results in slower, but more effective, training.

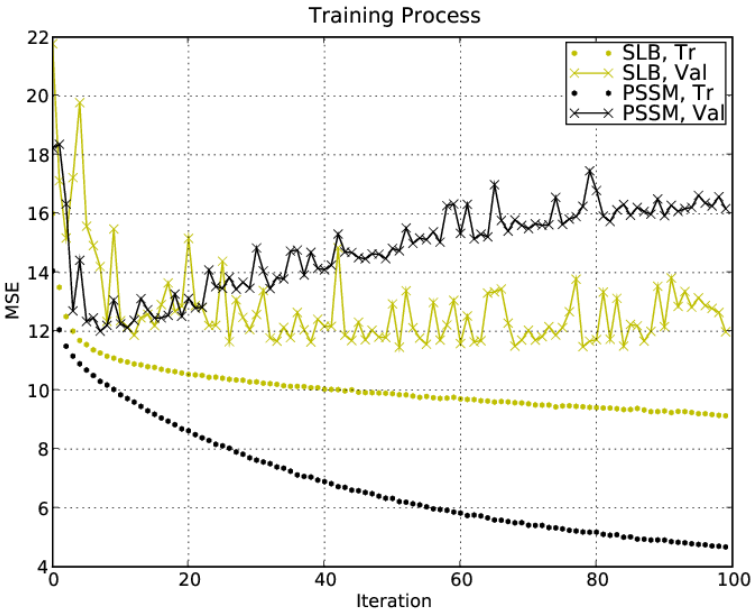


Figure 5.7: Comparing the behaviors of SLB and PSSM encoding methods during the training activity.

Chapter 6

Exploiting Intra-Structure Information in Secondary Structure Prediction

Output encoding determines *what* the system is currently predicting, and is the primary way to incorporate intra-structure information in a predictor. Once a protein sequence is annotated, several choices can be made about the actual encoding. As for primary structures, also secondary structures can be represented by a profile, using position-independent and position-specific information. The most common specific-independent encoding uses a one-hot representation, with $H =< 1, 0, 0 >$, $E =< 0, 1, 0 >$, and $C =< 0, 0, 1 >$. This encoding will be called HEC hereinafter. With this choice, a real-valued prediction can be decoded by choosing the label with maximum value. Supposing that values in fact estimate the probability of the residue to belong to that specific secondary structure class, the class chosen with this criterion maximizes the a-posteriori probability (APP) of a residue to belong to (MAP criterion). It has been statistically proven that predictors based on ANNs are in fact a-posteriori probability estimators when used as classifiers[1], and also the output of algorithms that are not naturally APP estimators can be remapped in the space of APP. For example, Platt *et al.* (165) proposed a heuristic mapping for support vector machines. In this case, obtaining a

[1] ANN estimate the a-posteriori probability of the classes they see during their training. Should the a-priori probability of the classes in the training set not be representing of the real distribution of the classes, it must be taken into account during the decoding phase.

measure of reliability for every single prediction in a simple way is also possible. The reliability can be expressed through a confidence value χ (also called reliability index), defined as follows (109):

$$\chi_i = \lfloor 10 * (max(\widetilde{V}_i) - next(\widetilde{V}_i)) \rfloor$$

where a) $\tilde{V}_i$ is a triple representing the output of a predictor at position i that is expected to approximate the adopted one-hot encoding, b) $max(\tilde{V}_i)$ and $next(\tilde{V}_i)$ denote the maximum and its closest value in the triple $\tilde{V}_i$.

In (110) was adopted an alternative position-independent coding for $\{H, E, C\}$, with $H =< 1, 0 >$, $E =< 0, 1 >$, and $C =< 0, 0 >$. According to the given definition, coil is implicitly asserted during a prediction when none of the values in the output pair exceeds a given threshold. Other position-independent alternative may be proposed exploiting the concept of code matrix (see Appendix 4.4.1). A notable encoding with position-specific information has been proposed by Petersen *et al.* (166), and considers three adjacent positions, i.e., a window of width 3 along an HEC profile.

In this chapter, two different ways to exploit intra-structure information through output encoding are presented. Firstly, with two different custom encodings with position-specific information are introduced, and two different combination techniques are proposed and experimented. Secondly, a novel method based on multi-faceted pipelines which use different encoding methods at different levels of the pipeline is presented.

6.1 Combining Heterogeneous Output Encodings

We here propose two alternative ways to encode secondary structures:

- *Transitions.* A two-bit representation that encodes transitions between zones belonging to different structures. The underlying semantics is the following: the first bit indicates an up transition (i.e., the current residue is the first residue of any structure) and the second bit a down transition (i.e., the current residue is the last residue of any structure). For example, the sequence CCECH would be represented with:

```
10111
01111
```

- *Beta-Segments*. A seven-bit representation that encodes the length of beta structure segments. Each bit corresponds to a range position: 1, 2, 3, 4-5, 6-7, 8-10, 11+. The bit of the corresponding range is set to 1. This coding tries to represent segments that have lengths which would make unlikely folding (beta segments are used to fold in groups that have similar length).

The proposed encodings are not by themselves "proper" output encodings, as they do not represent reversible transformations. They can be made reversible by adding the lacking information. The straightforward way to do that is to append the HEC coding to the representation of a residue. Preliminary experiments have been made to assess the performance with these techniques. Since no direct improvement was seen from the direct application of the proposed techniques, we proposed to combine them in order to exploit the possible heterogeneity in the predictions. To our best knowledge, no attempts have been proposed in combining heterogeneous output encoding techniques, therefore proper combining methods need to be devised. Two combination approaches are presented and experimented: transducer- and stacking-based.

6.1.1 The Proposed Combination Approaches

The former approach, depicted in Figure 6.1, adopts refiner experts as transducers devised to convert any output encoding into a final common encoding (HEC). Although

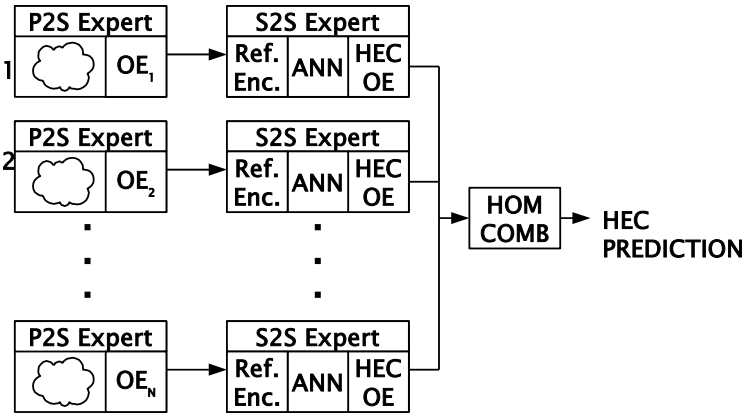


Figure 6.1: Transducer-based output combination

the combination is actually homogeneous, the original information should be preserved

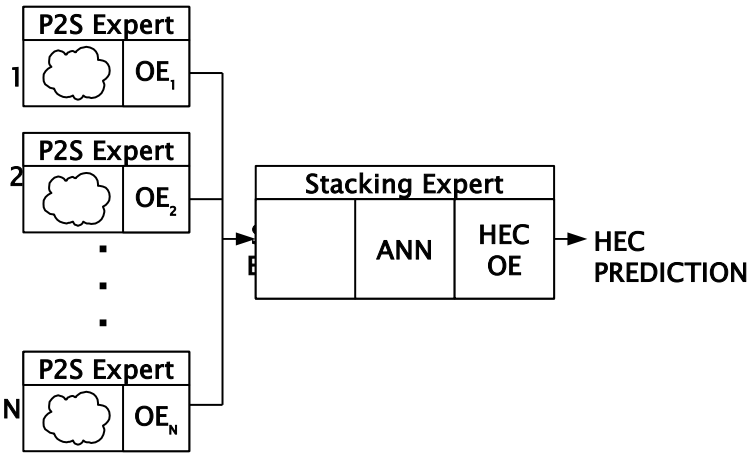


Figure 6.2: Stacking-based output combination

after the refinement step. The latter approach, depicted in Figure 6.2, adopts the stacking output combination technique to heterogeneous outputs. Let us recall that, as far as stacking is defined, a learner is trained on the outputs provided by the set of experts involved in the predictions. The stacking combination does not require homogeneous inputs nor semantics to be respected.

6.1.2 Experimental Results

Experiments have been performed comparing the proposed heterogeneous output combination techniques with homogeneous combination and the single experts. Five neural experts for every encoding from HEC, Transitions (TR), and Beta-Segments (BS) have been trained, resorting to a five-fold cross-validation process on 3500 independent proteins from nrDSSP dataset (127). The remaining 425 proteins are used as test set (T-nrDSSP, hereinafter). Main experts and refiners are defined as in 4.3.5.1, with only the Output Encoder module changing. Finally, 120 hidden neurons are used for the stacking expert. T-nrDSSP dataset has been used for testing to ensure that no related proteins are shared with training sets, and statistical significance is ensured by the reasonable dimension of the dataset (about 9% of the unrelated proteins available).

Table 6.1 show the results as follows: the first three rows represent the average results of single experts for the different encodings, after the refinements activity; the subsequent three rows represent homogeneous combination results; and the last two

rows represent the results of the proposed combination techniques.

System	Q_3	SOV	Q_{3_h}	Q_{3_e}	Q_{3_c}	C_h	C_e	C_c
HEC (single avg.)	79.38	77.15	85.05	69.52	79.40	0.738	0.664	0.604
TR (single avg.)	79.22	77.01	83.20	69.18	80.83	0.739	0.659	0.603
BSEG (single avg.)	79.51	77.31	83.77	70.02	80.59	0.741	0.666	0.607
HEC (hom. comb.)	80.11	78.08	85.53	70.44	80.26	0.749	0.677	0.618
TR (hom. comb.)	80.19	78.07	83.83	70.20	82.08	0.752	0.677	0.620
BSEG (hom. comb.)	80.18	78.03	84.15	70.54	81.59	0.751	0.678	0.620
Transducer	80.33	78.18	84.64	70.51	81.53	0.753	0.679	0.622
Stacking	79.88	77.41	84.04	70.41	81.02	0.744	0.669	0.614

Table 6.1: Results for dataset T-nrDSSP

Results show a slight increase in performance while using the transducer-based combination technique compared to the homogeneous combinations. Performances do not appear equally promising for the stacking-based technique. It is worth considering that, in the case of stacking, the combination is done without other refinement modules.

6.2 Exploiting Intra-Structure Information with Multi-faceted Pipelines

We here present an abstract software architecture –called SSP2– based on the combination of pipelines in which the different levels differ for the kind of encoding used. Then, a suitable predictor –called GAMESSP2– has been implemented and tested with the use of the framework GAME, compliant with the SSP2 abstract architecture. GAMESSP2 implements output encoding variation by changing the number of around residues considered in the representation of a secondary structure element at different levels of the pipeline.

6.2.1 Introducing the SSP2 Architecture

The SSP2 architecture extends the PHD architecture pointing to the most effective ways for exploiting the intra-structure correlation information. Forcing the output coding to add information about the surrounding of a residue is a way to obtain this desirable effect. This operation should be dealt with carefully, as adding too-specific information tends to make the learning process more difficult. We propose to introduce the intra-structure information step-by-step, at different levels of a pipeline. Adopting a transducer-based combination policy, previously described in Section 6.1.1, allows to put together systems which adopt different output encoding alternatives.

According to these assumptions, the SSP2 architecture specifies a set of constraints to be fulfilled by any compliant system:

- *ensemble.* At the most abstract view, a secondary structure predictor consists of an ensemble of pipelines. Each pipeline is expected to output its own prediction, with a degree of reliability that depends on the overall performance of the pipeline –measured on a suitable validation set. The choice about which blending policy should be adopted is left to the designer, jury decision being the default.

- *pipelines.* Each pipeline is expected to be defined in accordance with a k-step iterative strategy, each step of the being dealt with by a suitable software module. Heterogeneous pipelines are permitted, provided that the pipeline is able to take into account intra-structure correlation (through a suitable choice for output encoding of at least one of its modules).

- *modules.* Each module in a pipeline may be characterized by a specific technology (such as ANNs or SVMs) and uses specific input and output encoding methods, as well as parameter settings.

Figure 6.3 shows two specific examples of pipelines: the first performs a 2-step unfolding and the second a 3-step unfolding (the encoding process, as well as intermediate profiles have been disregarded for the sake of simplicity). It is worth pointing out that the 2-step generic pipeline resembles the P2S-S2S pipeline reported in Figure 3.2 for PHD, the main difference being that PHD does not use information about the primary sequence at the second step. Alternative pipelines are feasible, depending on the

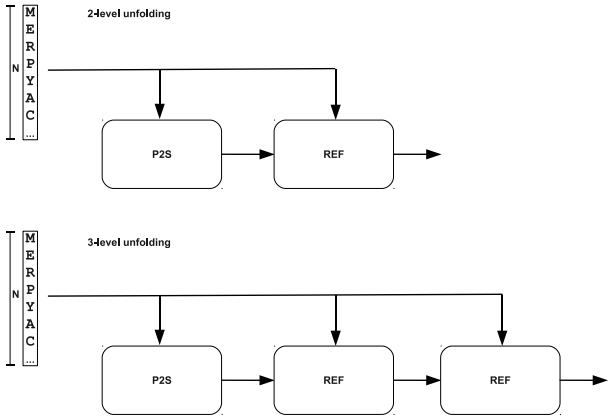


Figure 6.3: An SSP2 architecture with 2- and 3-levels of unfolding.

number of steps and on the kind of encodings one decides to apply.

To give additional degrees of freedom while promoting diversity, changes in the actual encoding and/or decoding methods are also permitted along the pipeline. In so doing, the possibility of making alternative implementation choices at each step gives rise to a great number of architectural alternatives. Since a thorough investigation of all systems compliant with the SSP2 abstract architecture is not feasible, we concentrated our efforts on the output representation, because it is less studied and thus more promising for further improvements.

Implementation (GAMESSP2)

GAMESSP2 is a realization of the SSP2 abstract architecture, implemented with GAME. The feasible variations of the SSP2 architecture being virtually infinite, we

concentrated our experiments on a significant subset for performance assessment. Implementing a system compliant with the SSP2 abstract architecture with GAME is straightforward. The first element of each pipeline is realized with a P2S Expert, further elements use a S2S Expert. The P2S and S2S modules, as well as the combination of them, have been implemented according to the standard setup shown in Section 4.3.5. Multiple alignments are obtained running PSI-BLAST on the data repository *uniref90filt*, with the inclusion threshold set to 10^{-3} and the number of iterations set to 3.

GAMESSP2 implements output encoding variation by changing the size of the output encoding window at different levels of the pipeline. The different benchmarking systems were built manually using the graphical interface of GAME. The Parameter *windowSize*, defined for all profile Encoders, was used to modify window size.

6.2.2 Results and Discussion

Tests have been performed with two main datasets, according to the guidelines stated in the context of the automatic assessment EVA (131). The former tests have the main purpose of assessing the performance among different SSP2 pipelines. These tests have been performed on the SD576 dataset, based on the SCOP database (22) definitions. The same 7-fold assessment proposed in (128) has been used to permit a fair comparison with the DBNN technique, which has shown to be very effective in the field of SSP. The latter tests have been performed on the EVA common 6 dataset (EVAC6), consisting of 212 proteins in total, divided into six subsets so as to include as many predictors as possible in the evaluation. The training set used in these experiments, i.e. EVAtrain, is composed of proteins with sequence identity less than 25% with any of the EVAC6 proteins.

6.2.2.1 SD576 Experiments (aimed at identifying best-performing pipelines).

Different pipelines (see Figure 6.4) have been tested and compared. In this setting, relevant parameters are the type of encoding (PSSM, SLB (3), or none) [1] and the length of the output window W_{out} (from 1 to 11). To avoid misunderstandings about

[1] Specifying the value *none* as a encoding method for a refiner, means that the refiner uses only the inputs provided by the previous expert in the pipeline.

which expert a window length refers to, by convention the parameters j and k will be used hereinafter to denote the length of the output window that refers to the first and the second expert in a pipeline (the output window of the third step has $length = 1$ in all experiments).

Figure 6.4 shows that two kinds of parameters are actually taken into account: the encoding method and the length of the output window. Experimental results are

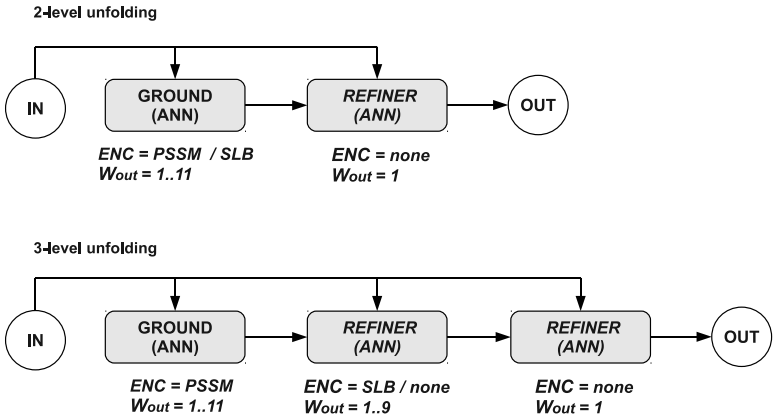


Figure 6.4: Two example pipeline configurations, with 2- and 3-level unfolding, parameterized using the encoding method (ENC) and the length of the output window (W_{out}).

reported according to the selected encoding method(s), while varying the length of the output window(s). In particular, four different kinds of pipelines have been tested:

- 2-step pipelines: a) $ENC =< PSSM, none >$ or b) $ENC =< SLB, none >$;
- 3-step pipelines: a) $ENC =< PSSM, none, none >$ or b) $ENC =< PSSM, SLB, none >$.[1]

Tables 6.2 and 6.3 report results obtained by running implementations of the above pipelines on the SD576 dataset, while varying the length of the output window. For each table, the best experimental results for Q_3 and SOV are highlighted in bold. It is worth noting that the SLB encoding worsens when enlarging the output windows. For this reason, it has not been used as a primary encoding method.

The four best-performing kinds of pipelines, in terms of Q_3 and SOV, adopt a three-step unfolding (the best results are in bold in Table 6.3):

[1]Recall that using "none" implies the absence of encoding for data inputs, i.e., the corresponding module actually disregards inputs coming from multiple aligment.

- *pipeline #1.* The first step uses PSSM and provides an output window of length $j = 3$, while the second step disregards inputs from multiple alignment and provides an output window of length $k = 7$ (i.e., $ENC =< PSSM, none, none >$ and $W_{out} =< 3, 7, 1 >$);

- *pipeline #2.* The first step uses PSSM and provides an output window of length $j = 5$, while the second step disregards inputs from multiple alignment and provides an output window of length $k = 7$ (i.e., $ENC =< PSSM, none, none >$ and $W_{out} =< 5, 7, 1 >$);

- *pipeline #3.* The first step uses PSSM and provides an output window of length $j = 3$, while the second step uses SLB and provides an output window of length $k = 7$ (i.e., $ENC =< PSSM, none, none >$ and $W_{out} =< 3, 7, 1 >$);

- *pipeline #4.* The first step uses PSSM and provides an output window of length $j = 5$, while the second step uses SLB and provides an output window of length $k = 7$ (i.e., $ENC =< PSSM, none, none >$ and $W_{out} =< 5, 7, 1 >$).

The four best-performing pipelined experts identified with the SD576 experiments have been combined (with jury decision) to give rise to the final GAMESSP2 configuration. The result for the 7-fold cross validation on the SD576 dataset, compared with the same test performed on DBNN, is given in Table 6.4.

6.2.2.2 EVA Common Set Experiments (aimed at performing benchmarking).

GAMESSP2 has been trained with the EVAtrain dataset, tested on the five EVA common sets,[1] and compared with different state-of-the-art algorithms. Tables 6.5, 6.6 and 6.7 report the results obtained by testing GAMESSP2 on EVA common sets.

The majority of experimental results report the best SOV for GAMESSP2. This noticeable result points to the ability of GAMESSP2 to take into account the intrinsic correlation that holds at the secondary step among amino acids that belong to the same secondary structure. This result experimentally confirms the conjecture that injecting information about the intra-structure correlation while performing error backpropagation permits one to obtain better results.

[1] See `http://cubic.bioc.columbia.edu/eva/sec/common3.html`.

System	Q_3	SOV	C_h	C_e	C_c
$j = 1$	79.66	76.68	0.75	0.67	0.61
$j = 3$	80.07	77.59	0.75	0.67	0.61
$j = 5$	80.18	78.12	0.76	0.67	0.62
$j = 7$	**80.20**	**78.12**	0.76	0.67	0.62
$j = 9$	79.99	78.02	0.75	0.67	0.62
$j = 11$	79.98	78.10	0.75	0.67	0.62

(a) $ENC =< PSSM, none >$, $W_{out} =< j, 1 >$

System	Q_3	SOV	C_h	C_e	C_c
$j = 1$	79.60	75.88	0.74	0.66	0.61
$j = 3$	**80.08**	77.98	0.75	0.67	0.61
$j = 5$	79.78	77.45	0.74	0.66	0.61
$j = 7$	79.46	**76.52**	0.74	0.66	0.61
$j = 9$	79.01	76.32	0.73	0.65	0.60
$j = 11$	79.01	76.32	0.73	0.65	0.60

(b) $ENC =< SLB, none >$, $W_{out} =< j, 1 >$

Table 6.2: Tests on SD576: Results for 2-step pipelines.

System	Q_3	SOV	C_h	C_e	C_c
$j=1,k=1$	79.70	77.31	0.74	0.67	0.61
$j=1,k=3$	79.71	77.46	0.75	0.67	0.61
$j=1,k=5$	79.79	78.00	0.75	0.67	0.61
$j=1,k=7$	79.80	78.14	0.75	0.67	0.61
$j=1,k=9$	79.78	77.70	0.75	0.67	0.61
$j=3,k=5$	80.17	78.37	0.75	0.67	0.61
$j=3,k=7$	80.14	**78.87**	0.75	0.67	0.62
$j=3,k=9$	80.02	78.21	0.75	0.67	0.61
$j=5,k=7$	**80.38**	78.45	0.76	0.68	0.62
$j=5,k=9$	80.13	78.55	0.76	0.67	0.62
$j=7,k=9$	80.23	78.65	0.76	0.67	0.62

(a) $ENC =< PSSM, none, none >$, $W_{out} =< j, k, 1 >$

System	Q_3	SOV	C_h	C_e	C_c
$j=1,k=1$	79.70	77.31	0.74	0.67	0.61
$j=1,k=3$	79.66	77.43	0.74	0.66	0.61
$j=1,k=5$	79.95	77.67	0.75	0.67	0.61
$j=1,k=7$	80.09	77.79	0.75	0.67	0.62
$j=1,k=9$	79.97	77.97	0.75	0.67	0.62
$j=3,k=5$	80.11	78.00	0.75	0.67	0.62
$j=3,k=7$	**80.34**	78.29	0.76	0.67	0.62
$j=3,k=9$	80.12	78.24	0.75	0.67	0.61
$j=5,k=7$	80.17	**78.78**	0.75	0.67	0.62
$j=5,k=9$	80.14	78.54	0.76	0.67	0.62
$j=7,k=9$	80.06	78.47	0.75	0.67	0.62

(b) $ENC =< PSSM, SLB, none >$, $W_{out} =< j, k, 1 >$

Table 6.3: Tests on SD576: Results for 3-step pipelines.

System	Q_3	SOV	C_h	C_e	C_c
$GAMESSP2$	81.22	80.03	0.77	0.69	0.63
$DBNN$	80.0	78.1	0.77	0.68	0.63

Table 6.4: Results for the best GAMESSP2 combination on the SD576 dataset.

System	Q_3	SOV	C_h	C_e	C_c
$PHDpsi$	73.37	69.54	0.64	0.68	0.52
$PSIpred$	76.79	75.44	0.67	0.73	0.55
$PROF_{king}$	71.63	67.74	0.62	0.68	0.51
$SAMt99_{sec}$	77.16	74.58	0.66	0.71	0.59
$PROF_{sec}$	-	-	-	-	-
$DBNN$	**78.80**	74.80	0.72	0.64	0.62
$GAMESSP2$	78.66	**78.09**	0.71	0.78	0.60

(a) Common set 1 (80 chains)

System	Q_3	SOV	C_h	C_e	C_c
$PHDpsi$	74.32	70.60	0.66	0.68	0.52
$PSIpred$	77.37	75.34	0.68	0.72	0.56
$PROF_{king}$	71.70	66.87	0.62	0.68	0.49
$SAMt99_{sec}$	77.17	74.53	0.66	0.71	0.57
$PROFsec$	76.21	74.91	0.67	0.71	0.55
$DBNN$	77.30	71.90	0.71	0.64	0.57
$GAMESSP2$	**77.62**	**75.65**	0.69	0.76	0.58

(b) Common set 2 (175 chains)

Table 6.5: Results for EVA common sets 1 and 2.

System	Q_3	SOV	C_h	C_e	C_c
$PHDpsi$	74.34	70.37	0.66	0.68	0.52
$PSIpred$	77.28	75.29	0.68	0.73	0.55
$SAMt99_{sec}$	77.10	74.35	0.66	0.71	0.56
$PROFsec$	76.40	74.93	0.67	0.71	0.56
$DBNN$	77.3	71.9	0.71	0.64	0.57
$GAMESSP2$	**77.69**	**75.66**	0.69	0.76	0.58

(a) Common set 3 (179 chains)

System	Q_3	SOV	C_h	C_e	C_c
$PHDpsi$	74.99	70.87	0.66	0.69	0.53
$PSIpred$	77.76	75.36	0.69	0.74	0.56
$PROFsec$	76.70	74.76	0.68	0.72	0.56
$SAMt99_{sec}$	-	-	-	-	-
$DBNN$	77.8	72.4	0.71	0.65	0.58
$GAMESSP2$	**78.34**	**76.17**	0.70	0.76	0.59

(b) Common set 4 (212 chains)

Table 6.6: GAMESSP2: Results for EVA common sets 3 and 4.

System	Q_3	SOV	C_h	C_e	C_c
$PHDpsi$	73.46	69.50	0.66	0.67	0.50
$PSIpred$	76.00	72.52	0.68	0.68	0.53
$PROF_{king}$	70.92	65.13	0.64	0.66	0.47
$SAMt99_{sec}$	76.38	73.13	0.67	0.68	0.54
$PROFsec$	75.43	73.25	0.69	0.69	0.53
$DBNN$	76.4	72.4	0.73	0.67	0.59
$GAMESSP2$	**76.95**	**73.27**	0.70	0.70	0.56

Table 6.7: GAMESSP2: Results for EVA common set 5 (73 chains).

6.2.3 Conclusions

A high accuracy for GAMESSP2 has been reported in standard tests performed on proteins characterized by no, or very far, evolutionary relationships with respect to those used for training. Comparisons with state-of-the-art predictors, made in accordance with the typical guidelines used for SSP performance assessment, put GAMESSP2 at the top of current state-of-the-art secondary structure predictors in almost all the experiments. In particular, the performance improvement is consistent with the recent advances in the field (about 1-2% in the last ten years). The best improvements have been obtained for the SOV measure: it can be explained with the capability of GAMESSP2 to take into account the intrinsic correlation within secondary structures. The experiments confirm the validity of the assumption that intra-structure correlation can be further exploited to improve predictions. The simple approach proposed, based on sliding windows, concentrates on local correlations; this suggests that further improvements could be achieved by devising output coding techniques able to exploit more distant relationships.

GAMESSP2 has been released as a web server and stand alone application. The main page is `http://iasc.diee.unica.it/ssp2`.

Chapter 7

Exploiting Inter-Structure Information in Secondary Structure Prediction

Correlation among secondary structures, evident when looking at homologous proteins, can be seen also in absence of clear evolutionary relationships. The space of actual folding alternatives is limited to some thousands despite the much greater variability observed in protein relationships (71). Tendencies in the formation of secondary structures can be evidenced looking at the distribution of structural features.

As shown in Figure 7.1[1], about 30% of helices have length 3 or 4, and while longer helices are in general less probable than shorter ones, finding an helix of length 10 is more probable than finding one with 5 residues. As for beta strands (Figure 7.2), single bridges constitute the most common beta structure type, and preference short segments are generally preferred, although strands of length 4 and 5 appear to be preferred to strands of length 2 and 3. Figures 7.3 and 7.4 show similar tendencies in the distribution of observed sheets within a protein and of the number of strand within a sheet, respectively.

These distributions, although extracted from inter-structure relationships, actually express "soft" intra-structure preferences, and may be exploited for SSP in the refinement or decoding phase. The S2S prediction modules of PHD-like predictors, although

[1]Distributions have been computed from a set of 8300 secondary structures of unrelated or distantly related proteins.

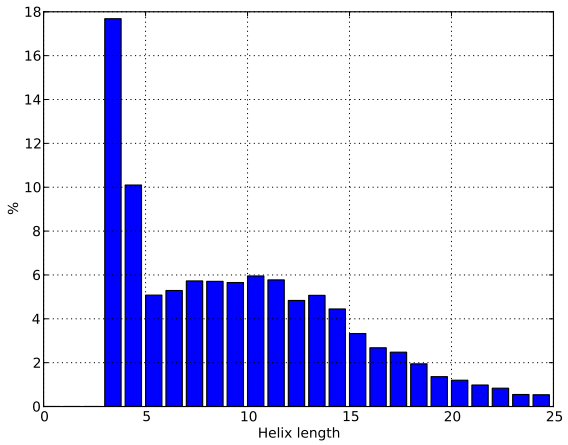


Figure 7.1: Observed length distribution of helices.

merely working at an intra-structure level, are able to take into account some local preferences in some way. We may say that this is due to the fact that the adopted learners can "see" the distribution of the examples used during their training process. Yao *et al.* (128) used the distribution of segment lengths to model the dynamic Bayesian network model used by their predictor. Following in this Chapter, the first results of an ongoing work about a completely novel proposal to exploit the information coming from structural feature distribution in the decoding phase is presented.

7.1 Decoding Secondary Structure Predictions

Structural classes are typically assigned by taking the maximum a-posteriori probability class independently for every residue, according to the estimations given by the underlying algorithm. Calling this criterion MAP_{local}, it is applied to single assignments ω_r to generate a predicted structure $\hat{S}$:

$$MAP_{local} : \arg\max_{\hat{S}} \hat{P}_{local} = \arg\max_{\omega_r} \hat{P}_r(\omega_r),\ \forall r \in chain, \tag{7.1}$$

where $\hat{P}_r(\omega_r)$ is the estimated a-posteriori probability for the class ω_r. A merely local approach cannot consider any intra-structural constraint: for instance, there is no way to take into account the distributions seen in Figures 7.1 and 7.2 with a MAP_{local} criterion, and even helices with length 1 are allowed.

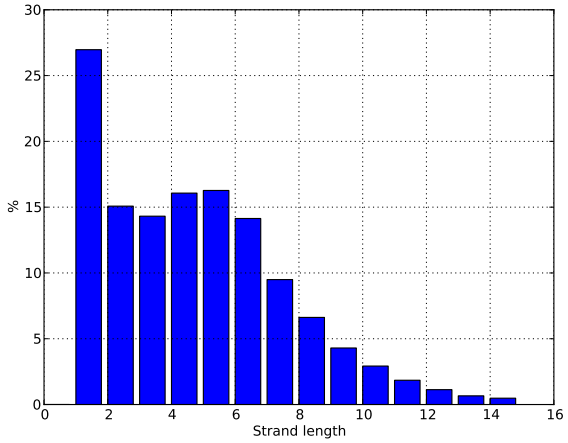


Figure 7.2: Observed length distribution of beta strands.

Should we be able to estimate the a-posteriori probability for a whole structure P_{global}, we may apply the MAP criterion to the full protein structure. Unfortunately, no algorithm in hand is able to give such as an estimation reliably enough to predict the structure. What we may concretely do is to search for inter-structure correlations that give us the statistical likelihood for a given structure to appear in a real protein. Even if this feeble estimation cannot be used to predict structure, it may be useful in combination with standard local techniques. Assuming that such obtained global estimations are independent from the local estimations given by a standard classification algorithm[1], the global MAP criterion may be expressed separating the two contributions:

$$MAP_{total} : \arg\max_{\hat{S}} \hat{P}_{total} = \arg\max_{\hat{S}} \left(\hat{P}_{local}(S) \cdot \hat{P}_{global}(S)\right) \tag{7.2}$$

Different interpretations may be given about the local probability formulation referred to the whole structure $\hat{P}_{local}$. A possibility is to consider the single estimations as independent:

$$\hat{P}_{local}(S) = \prod_{r \in chain} \hat{P}_r(\omega), \tag{7.3}$$

[1]Independence may be a very strong approximation in some cases, as global and local estimations asymptotically approach one each other as prediction improves. On the other hand, in this specific context we are more interested to not very good predictions.

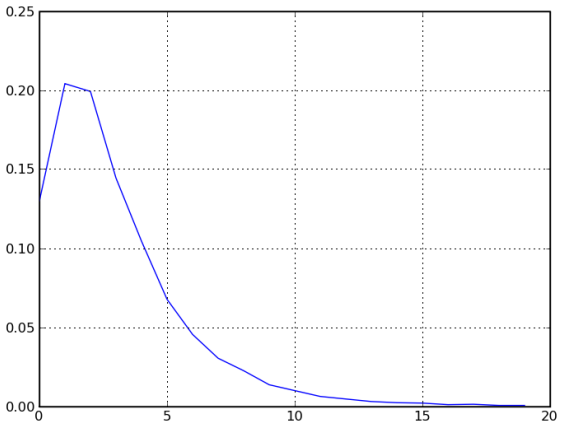


Figure 7.3: Observed distribution of the number of beta sheets within a protein.

The probability maximization may be obtained by using a search algorithm, moving around the prediction given by the MAP_{local} criterion. In other words, the idea is to perturb the initial annotated solution around the less safe local estimations.

Calling $\hat{S}$ the prediction obtained with MAP_{local}, the problem can then be formulated with the maximization of the probability for a structure $\hat{S}'$ around $\hat{S}$:

$$MAP_{total} : \arg\max_{S'}(\hat{P}_{local}(S' \mid \hat{S}) \cdot \hat{P}_{global}(S' \mid \hat{S})) \tag{7.4}$$

The conditional local probability for a structure element r can be expressed as the difference for the probabilities for the classes ω and ω' for that position, as assigned respectively in S and S':

$$\hat{P}_r(\omega' \mid \omega') = \hat{P}_r(\omega) - \hat{P}_r(\omega'), \tag{7.5}$$

As for the global term, to exploit the statistical properties of the space of structures is necessary to first detect and measure them.

The usual sequential representation of secondary structure does not permit to detect statistical properties usable to detect inter-sequence correlations. SS sequences can differ both in length and composition: also highly similar structures are distinguishable in a such defined space of structures. Considering the intersection between the space $\mathcal{S}$ of real structures and the space of predicted structures $\hat{\mathcal{S}}$, only a handful of points of contact could be seen between the two. This does not mean that predictions are not

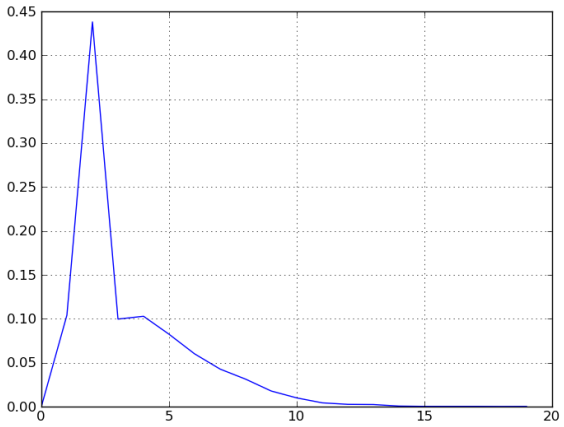


Figure 7.4: Observed distribution of the number of beta strands within a sheet.

significant: it rather shows that one-to-one comparison with the chosen representation is not able to detect any structural similarity in absence of an exact (although significant) matching. Adopting a soft-match criterion, for example using a threshold on structure sequence alignment percent identity, could appear as a possible solution. Although such criterion would increase the overlap between the two spaces, it would not permit to extract a distribution usable to estimate P_{global}.

A possible solution is to change of the space of representation, by defining a transformation $T : \mathcal{S} \rightarrow \mathcal{R}$ mapping a structure S into a representation R suitable for extracting statistically relevant features (Figure 7.5.

An ideal transformation T should highlight the statistical differences of interest while reducing the space. That would permit to experimentally extract and compare the distributions of predicted and real structures with sufficient reliability. This qualitative definition can be expressed quantitatively considering the properties of superposition of the distributions for $\mathcal{R}$ and $\hat{\mathcal{R}}$, obtained applying the chosen transformation to $\mathcal{S}$ and $\hat{\mathcal{S}}$ respectively.

Once defined $D_{T,\mathcal{S}}$ as the the distribution resulting from a transformation T on the space $\mathcal{S}$, a good transformation T should satisfy the following soft metrics-conditions:

- M_{pd} – *Prediction-diversity:* $D_{T,\mathcal{S}}$ should be different from $D_{T,\hat{\mathcal{S}}}$. This is representative of the fact that the distribution T is able to highlight prediction errors.

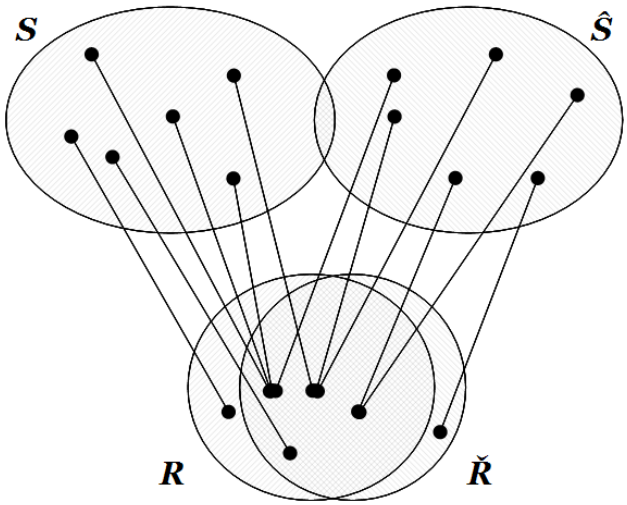


Figure 7.5: Example ideal mapping from the space of secondary structures to an alternative space.

- M_{ss} – *Self-similarity*: $D_{T,\mathcal{S}_i}$ should be similar to $D_{T,\mathcal{S}_j}$, where $\mathcal{S}_i$ and $\mathcal{S}_j$ are any two different realizations of the same space. Self-similarity is needed to reliably estimate the distribution.

These principles are in fact complementary: the former requires the important information related to the structure to be preserved. The latter requires the structure space to collapse into a smaller, abstract space.

Given a transformation T with the desired characteristics, the global term $\hat{P}_{global}$ from Equation 7.4 can be approximated in the transformed space considering the divergence between the distributions observed in the space of the predicted structures $\hat{\mathcal{S}}$:

$$\hat{P}_{global}\left(S' \mid \hat{S}\right) \sim \hat{P}\left(T(S) \mid T(\hat{S})\right) - \hat{P}\left(T(S') \mid T(\hat{S})\right). \tag{7.6}$$

Distributions can be estimated experimentally by enumerating the representations obtained for actual sets of predictions and real structures. The next steps consist of finding the proper transformation setting up the proper search algorithm.

7.2 Evaluating Structure Transformations

A good transformation should be able to evidence the global errors in a local predictor's estimation.

Manually determining a good representation which fulfills both the prediction-diversity and self-similarity conditions is not an intuitive task; the approach here proposed consists of generating a large set of different transformations and selecting the most promising ones according to the adopted criteria.

Both (prediction-diversity and self-similarity) conditions can be verified adopting a similarity measure between distributions. Similarity between two distributions can be measured by their euclidean distance:

$$d(D_1, D_2) = \sum_i (D_{1_i} - D_{2_i})^2, i\ D_1 \vee i\ D_2. \tag{7.7}$$

Then, prediction-diversity can be evaluated with the euclidean distance between the distributions of predicted structures $\mathcal{S}$ and real structures $\hat{\mathcal{S}}$. Analogously, self-similarity can be evaluated with the euclidean distance between the distribution between complementary subsets of $\mathcal{S}$ (lower the distance, higher the self-similarity)[1].

7.3 Determining Structure Transformations

In order to obtain our set of transformations we can eliminate or relax some constraints or definitions from the starting representation. This is coherent with the self-similarity criterion, which requires a representation to be more abstract than the original SS representation. Relaxing constraints allows originally distinct structures to merge into the same representation. The following constraints/definitions are considered:

- Residue position.
- Segment length: The length can be omitted (i.e. leaving only the alternation of segments) or discretized (e.g. long, medium, short segments can be grouped together).

[1] To be rigorous, the distribution for all the possible different couples of subsets should be considered, but a sensible number of instances should permit to give a proper estimation of the distance.

- Structural features: the number of symbols can be increased by identifying similar structural features.

The following representations have been identified (in order of increasing abstraction):

- *Segment sequence and length (SSL):* represents the structure as a sequence of segments. E 'perfectly equivalent to the representation of departure. *Example:* $EEEEHHH \rightarrow E4H3$.
- *Segment sequence (SEG):* as SSL, omitting the length.
- *Segment sequence – no coils (SEG-C):* as SEG, omitting coil segments.
- *Segment count (SC)*: count the segments of the same type, so giving a triplet.
  Example: $CCCCHHHHHHHCHHCCEEECCECCCCHHHHEECCCC \rightarrow (3, 5, 3)$
- *Segment count normalized (SCN)*: as SC, divided by the total number of residues, and discretized in 0-100.
  Example: $CCCHHHCCHCCHCECEECHHCCC \rightarrow (17, 26, 8)$
- *Segment count Discretized length (SCNx)*: same as SCN, discretized in 0-x
- *Segment count segment normalized (SCSN)*: as SC, divided by the total number of segments, and discretized in 0-10.
  Example: $CCCHHHCCHCCHCECEECHHCCC \rightarrow (3, 5, 1)$
- *Grouped H segments (GSH):* starting from SS-C, all sequences of consecutive segments of type H are compacted into a single H.
  Example: $CCCCHHHHHHHCHHCCEEECCECCCCHHHHEECCCC \rightarrow HEEH$.
- *Grouped segments (GS):* starting from the SS-C, all sequences are compacted segments of the same type.
  Example: $CCCCHHHHHHHCHHCCEEECCECCCCHHHHEECCCC \rightarrow HEHE$.
- *Segment number (SN):* counts the number of segments.

- *Segment number normalized (SNN):* the number of segments is divided by the total number of residues.
- *Segment number Discretized (SNx):* Number of segments discretized into x values.

7.4 Results

The cited metrics M_{pd} and M_{ss} have been assessed for all the specified transformations, using a euclidean distance measure for realizations of the distributions in the transformed space (Equation 7.7) .

M_{pd} can be calculated by computing the euclidean distance on the distributions of secondary structures and predictions obtained from a set of 8300 proteins with 80% or less sequence identity, say PDB80. The dataset was obtained using the PISCES culling server (167), and predictions were calculated with the predictor GAMESSP2 (see Chapter 6).

As for the self-similarity measure, it has been estimated with:

$$M_{ss} = \lim_{K\to\infty} \frac{1}{K} \sum_{k} d(D_{r,\mathcal{P}_k}, D_{r,\mathcal{P}'_k}) \tag{7.8}$$

$\mathcal{P}_k$ and $\mathcal{P}'_k$ being two random complementary realizations of the same protein set. The different datasets have been obtained by randomly shuffling the PDB80 and dividing each time the set in two subsets $\mathcal{P}_k$ and $\mathcal{P}'_k$. With k set to 5, no significant variation under different runs was seen for the metric value.

Table 7.1 shows the resulting values for the M_{pd} and M_{ss} in the transformations identified in Section 7.3. The transformations SCN, SCN5, SCSN, GS, SNN gave the best result.

Preliminary experiments have been made with these transformations with a weighted A* search algorithm (168). No relevant improvements have been still obtained with the use of the cited transformations.

7.5 Perspectives

After discussing the evidence in the presence of inter-structure information not related to the presence of homologues to be used as templates, a completely original proposal to deal with this kind of information in the decoding of a standard prediction has

Representation	M_{ss}	M_{pd}	Δ
SSL	0.999320680984	1.0	0.000679319015631
SEG	0.879753177584	0.842547770701	-0.0372054068836
SEG-C	0.862600480621	0.815796178344	-0.0468043022771
SC	0.527681838788	0.468535031847	-0.0591468069411
SCN	0.155341798121	0.462165605096	0.306823806975
SCN5	0.0612574559351	0.227515923567	0.166258467632
SCSN	0.0378466115145	0.167133757962	0.129287146447
GSH	0.720115891876	0.643566878981	-0.0765490128952
GS	0.0955727163881	0.165605095541	0.0700323791533
SN	0.238241220652	0.175541401274	-0.0626998193782
SNN	0.028792515426	0.24127388535	0.212481369924

Table 7.1: Result for the metrics M_{pd} and M_{ss} on the defined distributions. The transformations with potentially useful information are evidenced in bold.

been presented. Then, the theoretical aspects related to the introduction of a global estimation the in decision process have been analyzed. That permitted to conclude that, at least for really wrong predictions, an optimization process could be devised accounting separately for the local estimation normally given by a standard predictor and a global estimation with inter-structure information. With the goal of finding a proper estimation for the global term, the adoption of a transformation from the original space of structures has been proposed. Two different metrics have been also presented in order to evaluate the capability of a transformation for the purpose of prediction.

A preliminary set of eleven possible transformations have been proposed and evaluated according to the proposed metrics, finding five possible candidates for the use inside an optimization algorithm. Unfortunately, we are still not able to assess with certainty the validity of the proposed approach. Only very preliminary results are available for the optimization part; finding the proper definition for the global part of the cost function from the estimated probability should permit to obtain consistent improvements in prediction. More effective transformation functions may be also identified with the help of an expert of protein structures.

Conclusions and Perspectives

Bioinformatics is an interdisciplinary field that requires to blend together the knowledge belonging to the vast fields of computer science and biology. Coming from an engineering background, the most difficult aspect was to become familiar with a field of science of which I only had some resemblances from high school biological studies (not my favourite subject, I must admit!).

During these years as a PhD student, I have approached this new field like a miner who seeks the light digging at the edges of the heading. With the first attempts with protein secondary structure, I realized that the problem, like the best games, was much harder than suggested by its simple definition. There was no answer to the thousands of questions coming up after every failed experiment. With the purpose of reducing the time passed between a failed experiment and the other, I developed a system to plug in and configure different modules and run the experiments. After some time, and with the need to study different problems related to prediction in bioinformatics, that system became part of the research itself, and thus had a name, GAME.

Currently, GAME is a general software architecture and framework able to support the design and the release of predictors, explicitly devised real world problems.

In this thesis, the preliminary concepts about bioinformatics have been firstly presented, and the problem of secondary structure prediction has been analyzed in depth. After describing GAME, different contributions to the field of secondary structure prediction have been presented, from the perspective of the different biological information sources that can be exploited in a learning system. The proposed techniques are based on the encoding and decoding of this information, together with custom software architectures.

Not all the researches I have made in these years have found their proper place in this thesis. Among all, let me cite two still open researches, the prediction of the

packing angles of antibodies and and the prediction of disease-causing single amino acid polymorphisms in human proteins. A novel method for enhancing sequence alignment in the presence of templates has been presented in the appendix.

Different perspectives are open beyond the research related to secondary structure prediction. In recent years, the interest in the field has lowered due to the lack of really important improvements, probably also exhausted by the plethora of alternative solution published anywhere during the nineties. Conversely, the problem of ab initio prediction is still very actual and there is about 8% of accuracy that can be gained somewhere for the three class problem, and much more for the eight class problem. Methods used for secondary structure prediction can be also adopted for many other more specific prediction problems related to protein structure, and prediction of sequences in general.

As for GAME, the framework may be exploited for many other different problems, and may be enhanced with the addition of new alternative plugins.

Appendix A

Using a Beta Contact Predictor to Guide Pairwise Sequence Alignments for Comparative Modelling

A.1 Introduction

There are numerous methods for performing structural alignment which often differ in the precise details of their results (e.g. CE(169), SSAP(170), STRUCTAL(171), DALI(172), MATRAS(173), VAST(174), SSM(175)). Since there are many different ways to superimpose two or more protein structures, if the proteins are not identical (or at least extremely similar), then there can be no single optimal superposition(176). For our purposes, we have chosen the alignment produced by SSAP as the gold standard, 'correct' alignment.

```
1ap2A0                        DIVMTQSPSSLTVTAGEKVTM
1igmH0 Sequence alignment     EVHLLESGGNL-VQPGGSLRL
1igmH0 Structural alignment EVHLLESG-GNLVQPGGSLRL
                                        ****
```

Figure A.1: An example of an SSMA found between 1igmH0 and 1ap2A0. The SSMA is indicated with asterisks.

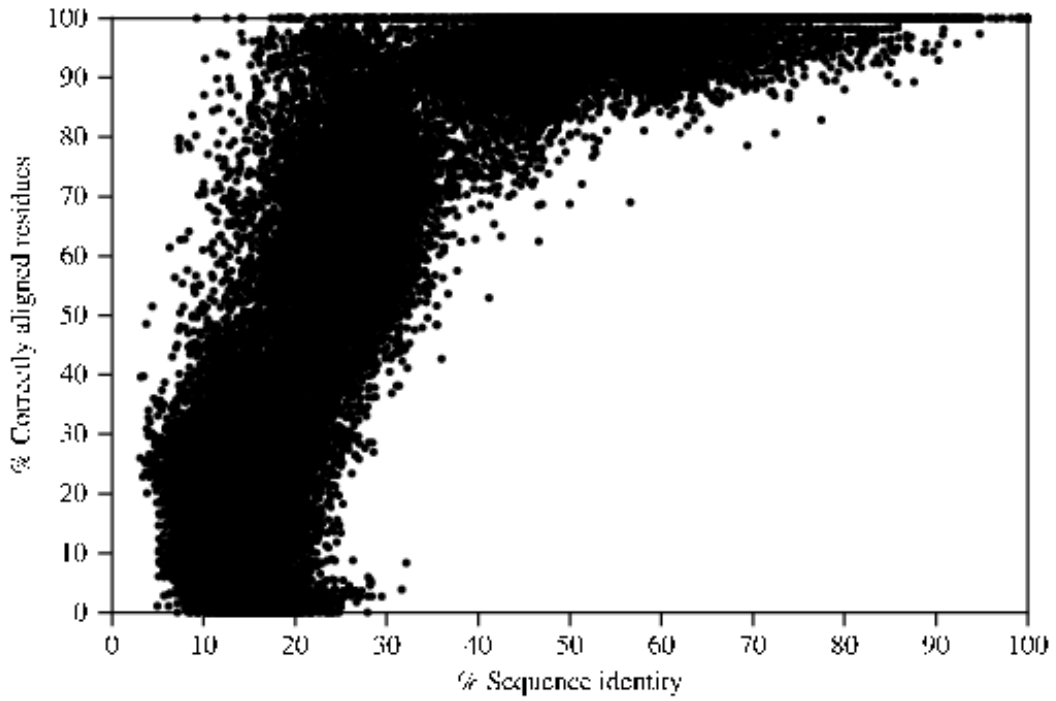


Figure A.2: The relationship between the percentage correct sequence alignment and the percentage sequence identity. Each pair of NRep domains in each CATH homologous family has been structurally aligned by SSAP and sequence aligned using a Needleman and Wunsch global alignment. The structural alignment is taken as the correct alignment. Twelve outlying points have been removed after being identified as occurring owing to errors in the CATH database.

Thus misalignment between a target and a parent sequence is the largest cause of error in comparative modelling. The most extreme types of misalignment (Misleading Local Sequence Alignments, MLSAs) are areas of protein alignment where structural similarity is clear and where the optimal sequence similarity is substantially higher than that seen in the structure alignment(177). In other words, the sequence alignment for a region is very clear, yet it does not match the structure-derived alignment. We define less extreme misalignments, where the sequence and structural alignments do not agree, as SSMAs ('Sequence-Structure MisAlignments'). For example, Figure A.1 shows the sequence and structural alignment of a region from 1igmH0 and 1ap2A0 (a human and mouse antibody heavy chain variable region respectively).

In their analysis of the CASP2 comparative modelling section, Martin *et al.*(69)

showed that there was a relationship between the percentage of correctly aligned residues and the sequence identity (Figure 2 of their paper). We have reproduced that analysis using approximately 56,000 pairs of homologous protein domains from CATH(178; 179), each of which was aligned on the basis of structure using SSAP and on sequence using a Needleman and Wunsch sequence alignment(26). Figure A.2 clearly shows that if there is a high sequence identity between two sequences, then the sequence alignment is likely to match the structural alignment. However as the sequence identity decreases, particularly below 30%, the accuracy of the alignment decreases and the sequence-based alignment can be completely different from that derived from the structural alignment. If we can predict regions where mis-alignment occurs then we can hope to improve the alignment in these regions and therefore improve the model. In this paper, we concentrate on improving the alignment in beta sheets.

Previous work by Fooks *et al.*(180), Lifson & Sander(181), Wouters & Curmi(182) and Hutchinson *et al.*(183) has shown clear residue pairing preferences between adjacent beta strands. With this in mind, we believe that some sequence mis-alignments can be detected and corrected by detecting errors in the assignment of beta contacts. In practice, given a pair of beta strands (which we refer to as a 'beta pair') assigned to a target from a template after initial sequence alignment, a measure of the likelihood of that pairing being formed in a real protein can be used as part of a scoring system of an alignment algorithm. On this basis, we have developed BCeval, a beta-contact evaluator based on a mixture of neural networks able to predict whether a pair of beta-strands is in the correct register. In addition, a pairwise sequence alignment method (BCalign) has been developed capable of automatically taking into account the beta contact evaluations while building the alignment. A search algorithm controlled by an iterative procedure is adopted to find the alignment instead of a classical dynamic programming technique such as Needleman and Wunsch. The score of a substitution in the alignment will also depend on the mutual register with another substitution along the sequence (because the register will affect the beta pairing), thus breaking the basic assumption of dynamic programming. In other words, while searching for the best alignment, the contacts of the template are assigned to the target; the scoring system then takes into account both the assigned beta strands at the same time, so that the substitutions within a strand cannot be scored without taking into account the information about the neighbouring strand.

A.2 Methods

When two sequences (the homology modelling target and template) are aligned, the structural characteristics of the template are assigned to the target. Thus the secondary structure and the relative position within the structure (including interactions with other residues) are immediately known for the target sequence. A mis-alignment (i.e. an incorrect substitution suggested by the sequence alignment) will lead to a wrong structure assignment: the bigger the shift from the correct position, the greater the distance from the correct structure. Thus we are able to examine contacts between residues in adjacent beta strands assigned in the target protein in an attempt to detect mis-alignments. In particular, we estimate the likelihood of an assigned beta pair being correct using an evaluator based on machine learning (BCeval).

At first glance, including these evaluations in the scoring system of a typical dynamic programming algorithm seems a straightforward way to use the new information. Unfortunately, the main dynamic programming assumption (that the optimal solution of the problems should depend on the optimal solution of its sub-problems) is broken. In order to overcome this limitation, we propose a technique which adopts a heuristic search algorithm (BCalign).

A.2.1 Developing the Beta Contact Evaluator (BCeval)

The evaluation of beta contacts can be tackled as a prediction problem, similar to contact map prediction. We must (i) define the training data, (ii) find a suitable representation of the input and output data and (iii) set up a proper architecture and learning algorithm(s). All the described modules have been implemented using the GAME framework(8), written in Java 6.0. Following the philosophy of GAME, the experiments and the combination of experts have been set up using the graphical interface. The resulting system is called BCeval.

A.2.1.1 Why not use a Generic Contact Map Predictor?

In principle, it might be possible to use a generic contact predictor to evaluate a contact between beta strands. Various contact predictors have been proposed, such as those by Cheng & Baldi(90) and Tegge *et al.*(91). However, the accuracy of these tools in not high, owing to the difficulty of predicting all possible contacts occurring in a

```
----AA-----AA------BBBB-----CCCCC-----CCCCC-------BBBB------
----12-----12------1234-----12345-----54321-------4321------
QSPVDIDTHTAKYDPSLKPLSVSYDQATSLRILNNGHAFNVEFDDSQDKAVLKGGPLDGT
CCCCEECCCCCEECCCCCCEEEECCCCCEEEEEECCCCEEEEECCCCCCCEEEECCCCCC
```

Figure A.3: An example chain indicating the residues in contact. The letters in the first line indicate the beta strand pairs. The numbers in the second line indicate the residues in contact within the same pair. For example, the two residues labelled B1 form a contact.

protein (including between α-helices). Even more specific predictors specialized in beta contacts report accuracies below 50%(184).

Fortunately, rather than the general problem of predicting contacts between β-strands, we already know which strands are in contact and we can concentrate on small shifts around a given position. Thus, we developed a new system specialized in recognizing a contact from the 'shifted' versions that could be identified from an alignment procedure.

A.2.1.2 Data Representation

The beta-pairs must be represented in a fixed-length vector in order to obtain an input suitable for a neural network. The main requirement for the input vector is to contain the residues of the two strands involved in the pairing. Additionally, the shifted versions of the same pair must be clearly recognizable.

Figure A.3 shows an example fragment of protein chain, in which the contacting residues belonging to different beta strands are indicated. Clearly the length of the β-segments is variable, while a fixed-length vector is needed for the data representation. Choosing to encode a window of N residues, that window would be perfectly suited to strands of length N, while information would necessarily be lost for pairings of longer strands. On the other hand, using larger windows would include residues which are not involved in contacts.

In addition, one must account for correct pairing of residues in both parallel and anti-parallel strands. For instance, taking a window of four residues along the anti-parallel B strands in Figure A.3, the data representation must indicate that the leucine at the first position in the first strand is in contact with the glycine in the last position of the second strand and not the valine in the first position. The different

hydrogen-bonding patterns observed in parallel and anti-parallel sheets also result in different propensities in the contacts between residues, as shown by Fooks *et al.*(180) and Hutchinson *et al.*(183). For these reasons, a 'mixture of experts' approach has been adopted: one expert deals with only strands of one type, without any adjustment in the encoding phase.

Profiles, obtained after three iterations of a PSI-BLAST search on the whole protein, have been used to encode the residues in the window. PSI-BLAST is run against the data repository *uniref90*[1], with the inclusion threshold set to 10^{-3} and the number of iterations set to 3. The default values of the NCBI stand-alone version of *blastpgp* were used for the remaining parameters.

A simple position-independent coding of the residues was also tried, but profiles lead to better performance.

A.2.1.3 The Architecture

The core of BCeval consists of a mixture of 13 neural networks, each one specialized for a specific length (1,2,3,4,5,6,7+) and type (parallel or anti-parallel) of beta pairing. Figure A.4 shows the architecture of BCeval. The 13 experts (bridge contacts of one residue do not distinguish between parallel and anti-parallel) are combined at the first level and constitute a 'core evaluation module'. The window length includes all (and only) the residues involved in each pairing, such that each neural network has a fixed-length vector as input, representing the residues involved in the contact. Simpler architectures with only one neural network and fixed input length (1, 2, 3) were tried first, but the described architecture improved accuracy. The final output is obtained by averaging three core evaluation modules trained separately.

A.2.1.4 Training data Composition

The composition of the training examples determines the ability of the trained machine learning system to fit the problem. A reference test set, TESTDOM, was built by selecting 10% of the total codes in the CATH database(21) at the homologue level, and taking the corresponding domains. A subset of the domains in TESTDOM, prevalently far homologues, have been used to build a set of pairs of domains to be aligned. The

[1] `http://www.pir.uniprot.org/database/nref.shtml`

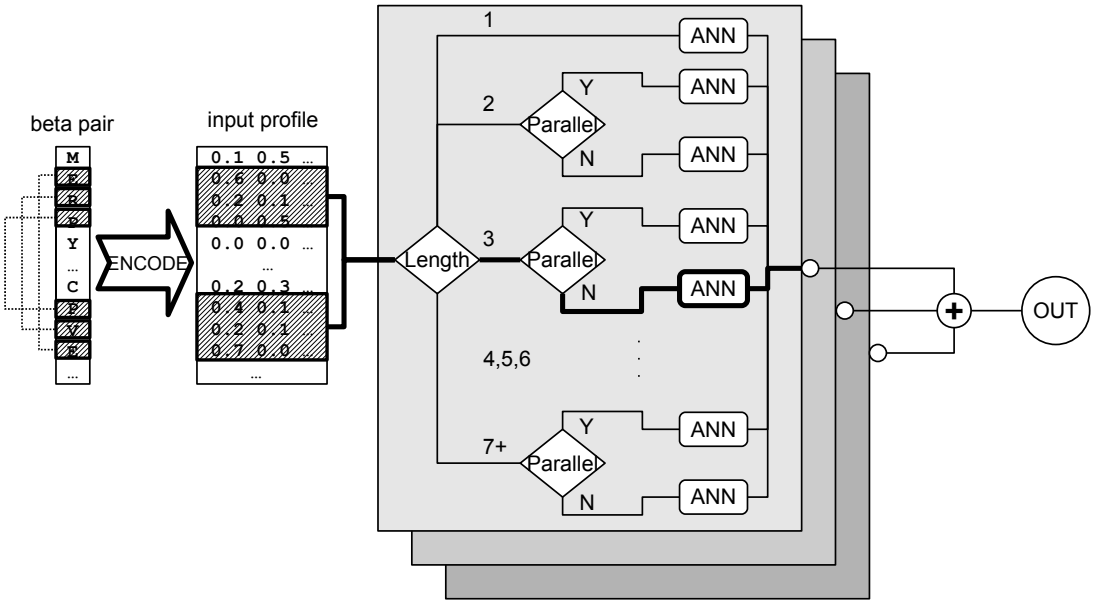


Figure A.4: The BCeval architecture. The guarding functions ensure that only one neural network is activated at a time. The 'parallel' guard is able to distinguish between parallel and anti-parallel strand pairs, while the 'length' guard dispatches based on the length. In the example, an anti-parallel pair of length 3 is given so activating the path shown in bold. Three core units consisting of independently trained neural networks are averaged to obtain the final evaluation.

resulting set, TESTALIGN, consists of 743 proteins, which have been used to test the alignment algorithms. A set of protein chains, TRAINCH, consisting of protein chains from a dataset with identity $< 25\%$[1], selected in order not to include any chain containing the domains in TESTDOM, was used as a starting point to obtain contacts; whole chains were preferred to domains in order to use DSSP(185) outputs from the EBI[2] directly. Contacts included in the training of BCeval have been obtained from within the chains of TRAINCH by parsing the DSSP files, which include the position of the contacts between pairing residues in β-strands. Negative examples (i.e. pairings not observed to be in a beta contact) have been obtained by a synthetic sampling around the actual contacts. A Gaussian distribution ($\mu = 0, \sigma = \sqrt{5}$) around the positive examples was used to perform the sampling. The negative and positive samples are balanced (half

[1] `http://bio-cluster.iis.sinica.edu.tw/~bioapp/hyprosp2/dataset_8297.txt`
[2] `ftp://ftp.ebi.ac.uk/pub/databases/dssp/`

positive, half negative), without taking into account the observed distribution. As seen in Figure A.2, the extent of the observed shifts depends greatly on the sequence identity, making it hard to model the observed distribution correctly and, in any case, balanced inputs generally result in better learning. This partial synthetic sampling has been preferred to sampling the real data in order to obtain more, and more varied, samples. In practice, the negative data are randomly generated at each training iteration, so improving the diversity given to the training algorithm.

A.2.1.5 Training Technique and Parameter Setting

Each expert is based on a 3-layer feed-forward neural network, trained with a variant of the back-propagation algorithm, with learning rate = 0.001 (initial value) and momentum = 0.1. The learning rate is adjusted between iterations with an inverse-proportion law. The number of hidden neurons is 75 for each neural network. The number of input neurons is $20 \cdot N$, where N is the size of the input window. A single output neuron indicates whether the given input is a contact or not. With the goal of improving the ability of the training algorithm to escape from local minima, the training set is randomly shuffled at each iteration, so that the learning algorithm is fed with the same set of proteins given in a different order. Furthermore, each protein provides only a subset of its inputs, according to a random choice performed in accordance with a specific parameter, n. In particular, a random value k is extracted in the range $[0, n-1]$ and the inputs with index k, $k+n$, $k+2n, \ldots$ are provided to the learning algorithm.

To prevent the training process from stopping with a local oscillation of accuracy (evaluated on a validation set consisting of a 10% of TRAINCH, not used in the back-propagation process), weights are recorded when a minimum is encountered on the validation set, but the training continues until the error on the validation set increases for 10 consecutive iterations.

A.2.2 Developing the Pairwise Sequence Alignment (BCalign)

The definition of an alignment algorithm includes two separate parts: (i) the cost function i.e. a scoring scheme used to evaluate an alignment; (ii) the alignment strategy, i.e. a strategy which gives the succession of substitutions, insertions and deletions which minimize the cost function. In the first part of this section, we describe a cost function

which includes the evaluation of beta pairings made by BCeval. In the second part, an alignment strategy suitable for use with the given cost function, is presented.

A.2.2.1 Defining the Cost Function

Given a pairwise sequence alignment, A, between a template and a target sequence (the structure of the template being known), its cost,[1] $c(A, S_{tpl})$, in BCalign consists of the sum of three main contributions (Equation A.1):

$$c_{bcalign}(A, S_{tpl}) = c_{nw}(A) + c_{betaindel}(A, S_{tpl}) + c_{bceval}(A, S_{tpl}) \tag{A.1}$$

where S_{tpl} is the structure of the template sequence.

The component c_{nw} is essentially the result of the application of a classical Needleman and Wunsch scoring scheme: the sum of the costs of all substitutions, insertions, and deletions occurring in the alignment. The cost of the substitutions is obtained from a similarity scoring matrix M_s (e.g. BLOSUM62), reversed so as to obtain a cost matrix $M_c = -(M_s - max(M_s))$. The cost of insertions and deletions is the same, and is described by an affine gap penalty: the cost of opening of a new gap (c_{gapop}) is greater than extending a gap that has already been opened ($c_{gap_{ext}}$):

$$c_{nw}(A) = c_{gap_{op}} \cdot n_{gap_{op}} + c_{gap_{ext}} \cdot n_{gap_{ext}} + \sum_{i,j} M_c(P_{tpl_i}, P_{tgt_j}) \tag{A.2}$$

where i and j indicate the position of the substituted residues, $n_{gap_{op}}$ the number of gap openings, $n_{gap_{ext}}$ the number of gap extensions, all according to the alignment A.

The term $c_{betaindel}$ in Equation A.1 is given by the total number of gaps within the beta strands in the template. Including this contribution is similar to the approach adopted in PIMA(186), with the exception that here only beta strands are included. This component has been included in order to increase the number of beta pairs available for the evaluation: the larger costs help to avoid insertions and deletions inside beta strands, which rarely occur during evolution. If n_{gap_β} is the number of gaps within beta strands, the cost is given by:

$$c_{betaindel} = n_{gap_\beta} \cdot c_{gap_\beta} \tag{A.3}$$

[1]Note that scores, usually preferred in the scoring systems of sequence alignments, can also be viewed as the opposite of costs.

where c_{gap_β} is β-specific gap penalty.

The last term in Equation A.1, c_{bceval}, results from the evaluation of beta pairs in the target sequence, assigned from the template. This has two different effects:

- to increase the cost of beta pairs that appear to be mistakenly assigned (i.e. shifted),
- to decrease the cost of beta pairs that appear to be assigned correctly.

The first of these changes the equilibrium of the alignment space, moving away from the solutions suggested by the other two terms that lead to wrong beta pair assignments. The second, although not directly permitting us to improve the alignment, prevents drifts when correct assignments are found with the standard scoring scheme. In fact, the change of a pair of assignments may affect the solution in many different places within the alignment.

Considering $\tilde{p}(bp_{tgt})$ as the estimation of the probability (in this case given by BPEVAL) of the beta pair bp_{tgt} being formed, it is reasonable that the cost $c(bp_{tgt})$ should be proportional to $-\tilde{p}(bp_{tgt})$. It is also reasonable that the value of the cost should be large enough to allow the overall cost function to escape from the misleading minima obtained with the standard scoring system. For this reason, the cost should also be proportional to the number of residues involved in the pairing (n_{bp}). After trying different options, the following quadratic formula appeared to give the best stable performance:

$$c_{ev_{abs}}(bp_{tgt}) = \begin{cases} (0.5 - \tilde{p}(bp_{tgt}))^2 \cdot n_{bp} & \text{if } \tilde{p}(bp_{tgt}) \leq 0.5, \\ -(0.5 - \tilde{p}(bp_{tgt}))^2 \cdot n_{bp} & \text{if } \tilde{p}(bp_{tgt}) > 0.5. \end{cases} \tag{A.4}$$

Considering the stochastic nature of the evaluator an extra term has been included, in order to stabilize the algorithm in the presence of wrong estimations. This extra contribution also takes into account the corresponding beta pair in the template (bp_{tpl}), for which the estimation error is known to be $1 - \tilde{p}(bp_{tpl})$. The requirement for this term derives from the assumption that the errors in the estimation of p on the template and the target are correlated. Therefore, in the presence of a confident prediction ($\tilde{p}(bp_{tgt}) > 0.5$), if $\tilde{p}$ for the template is significantly larger than for the target, we have a strong indicator of a probable error in the alignment. This relative contribution is given by:

$$c_{ev_{rel}}(bp_{tgt}, bp_{tpl}) = \begin{cases} (\tilde{p}(bp_{tpl}) - \tilde{p}(bp_{tgt}))^2 \cdot n_{bp} & \text{if } \tilde{p}(bp_{tpl}) - \tilde{p}(bp_{tgt}) > 0.1 \\ 0 & \text{otherwise.} \end{cases} \quad \text{(A.5)}$$

The total cost for a single beta pair is then:

$$c_{ev}(bp_{tgt}, bp_{tpl}) = \gamma_{abs} \cdot c_{ev_{abs}}(bp_{tgt}) + \gamma_{rel} \cdot c_{ev_{rel}}(bp_{tgt}, bp_{tpl}) \quad \text{(A.6)}$$

where γ_{abs} and γ_{rel} are given as parameters.

The total cost for a given alignment is given by the sum of c_{ev}, for all the beta pairs resulting from the assignment of the template structure S_{tpl} to the assigned target structure $\tilde{S}_{tgt}$:

$$c_{bceval}(A, S_{tpl}) = \sum c_{ev}(bp_{tgt}, bp_{tpl}), \forall bp_{tpl} \in S_{tpl}, bp_{tgt} \in \tilde{S}_{tgt} \quad \text{(A.7)}$$

A.2.2.2 Minimizing the Cost Function

Dynamic programming techniques, such as Needleman and Wunsch, are generally adopted to minimize a cost function for sequence alignments. These algorithms are the best choice when the best solution for the whole problem can be built incrementally by calculating the best solution for its sub-problems. When adopting the proposed cost function, this assumption is broken, since the cost of a substitution is related to other substitutions along the sequences. A natural generalization of the Needleman and Wunsch algorithm is represented by search algorithms, which allow us to evaluate the path dynamically. A search algorithm is completely defined by:

- *The search problem.* This is defined by the tuple *(S_0, operator-set, goal-test, f)*, where S_0 is the start state; *operator-set* defines the set of states that can be reached from a given state; *goal-test* can say whether a given state is the goal or not; f is an evaluation function which gives a score (or a cost) for a given path (sequence of states).

- *The search strategy.* This determines the order in which the nodes are expanded. In global search strategies, a node corresponds to a state in the tree. For instance, a best-first strategy always expands a node with the best value of f.

In other words, using a global-search algorithm, the best alignment can be found by searching for the path in a tree with minimum f, leading to the end of the sequences, in which each node of the tree is generated by the operations of insertion, deletion, or substitution. Figure A.5 gives an example of a simple alignment, equivalent to

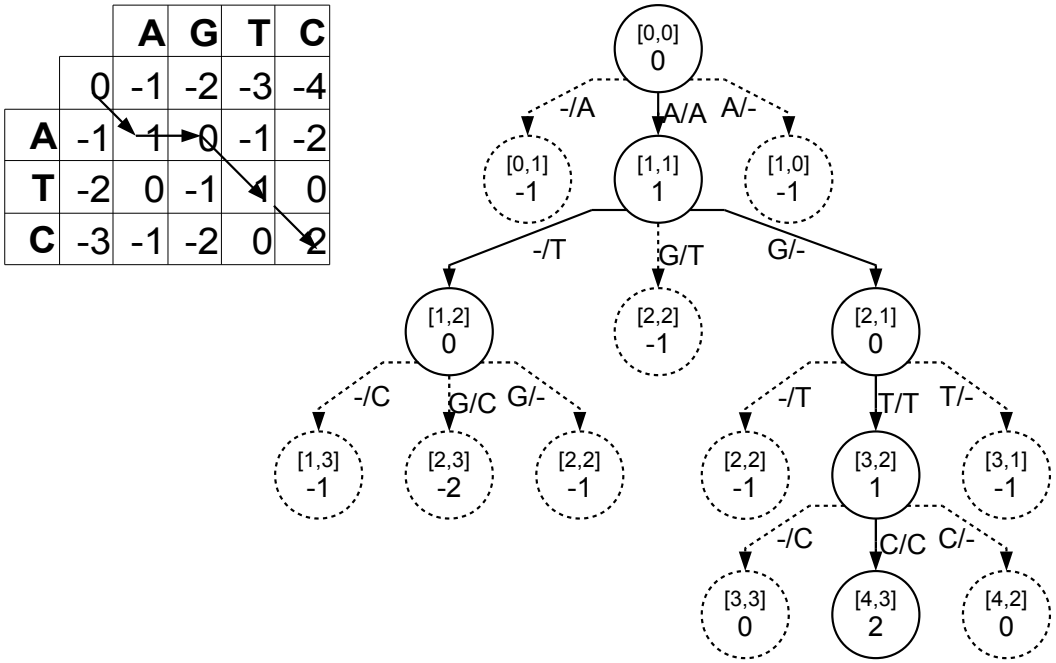


Figure A.5: An example of alignment performed with a search algorithm. The search can be represented by a tree, in which the edge score is given by a simple scoring system (-1 for gaps, 1 match, -2 mismatch). Each circle represents a node, indicating the position in the two sequences and the path score. With a best-first search (i.e. the most promising nodes are opened first), the nodes shown with solid lines are expanded. Also nodes outside the solution path (in dashed lines) are explored, according to the local score. On the left, the corresponding Needleman and Wunsch matrix is indicated: note that the values in the Needleman and Wunsch matrix correspond to the scores of a node only when the best path to that node is followed.

Needleman and Wunsch, performed with a best-first search strategy.

The pay-off for the greater flexibility of search algorithms is a possible explosion of the computational cost. For instance, in the example of Figure A.5, a blind (brute-force) search strategy is adopted, with the consequence that many nodes could be expanded before ending with the solution. The expected number of expanded nodes

grows exponentially with the length of the path, which in turn grows linearly with the length of the sequences. The number of expanded nodes is greatly reduced by adopting a heuristic search algorithm (for example, A*(168)). In A* the path cost of a given node n is the sum of two terms:

$$f(n) = g(n) + h(n) \tag{A.8}$$

g being a path cost function, and h being a heuristic function, expected to estimate the cost from that node to the solution. If the cost increases monotonically along the path and the heuristic function is admissible (i.e. it is an underestimation of the real cost of the solution) the A* algorithm is guaranteed to find the path with minimum cost. The complexity becomes linear if the estimation given by the heuristic function is exact. Using the cost function defined in the previous section as $g(n)$, a heuristic function can be devised able to estimate exactly the first two terms of the cost. Unfortunately, an exact estimation cannot be given for $c_{bceval}(A, S_{tpl})$, so that the complexity rapidly increases with the length of the sequences to be aligned. In order to control the run time, rather than dynamically apply the evaluator during the search, an iterative procedure has been used: at the *i-th* iteration, the pairings found in all $i-1$ previous iterations are evaluated, until a reasonable trade-off is found.

```
def bc_align(template_seq, target_seq, template_str):
    c_pairs=set()
    for x in range(MAX_ITERATIONS):
        solution = bc_search(template_seq, target_seq, c_pairs)
        new_c_pairs = c_pairs + get_c_pairs(solution, template_str)
        if len(new_c_pairs) > len(c_pairs):#no new pair found
            c_pairs = new_c_pairs
        else:
            break
    return solution
```

Listing 2: The iterative algorithm pseudo-code, in Python-like syntax.

Listing 2 presents the pseudo code for the iterative procedure. The function *bc_search* performs a search, applying the costs for the pairs obtained for the

beta-pairs encountered in the solutions of previous iterations. The iterations continue until convergence, or the maximum number of iterations is reached.

The A* evaluation function has been used in the alignment search algorithm, with the Iterative-Deepening A* (IDA*)(187) search strategy, which has been preferred for its better use of memory. In accordance with the example in Figure A.5, a search state is indicated by $S = [i, j]$. i and j represent the relative position in the sequences, and can also be viewed as coordinates in a Needleman and Wunsch-like cost matrix. Given the definition of the search state, the heuristic search problem is given by:

- S_0 **(start state):** $[0, 0]$
- **goal-test:** $[i_0, j_0], i_0 = length(P_1) \wedge j_0 = length(P_2)$
- **operators:** $[i_0, j_0] \rightarrow \{[i_0 + 1, j_0 + 1], [i_0 + 1, j_0], [i_0, j_0 + 1]\}$ (substitution, insertion, and deletion respectively). Each operation is defined provided that $i <= length(P_1) \vee j <= length(P_2)$.
- g **(cost function):** depends on the path from the start state to the current state. It includes the costs for the substitutions and gaps along the path, and the costs arising from beta-pair evaluations. The cost function is basically the $c_{bcalign}$ as defined in Section A.2.2.1; for performance reasons, the values are rounded to the nearest integer.
- h **(heuristic function):** the heuristic function is the estimated cost for the remaining part of the sequences from the current state. A matrix H_{P_1,P_2} gives the estimation: $h([i, j]) = H_{P_1,P_2}(i, j)$. The matrix H is constructed similarly to a Needleman and Wunsch matrix, minimizing the sum of the terms c_{nw} and $c_{betaindel}$.

A.3 Results

A.3.1 BCeval

Table A.1 shows the accuracy, specificity, and Matthews correlation coefficient(133) obtained after a 7-fold cross validation experiment on the dataset TRAINCH.

Tests were performed in order to evaluate the usefulness of BCeval in the evaluation of alignments, i.e. the possibility of using metrics obtained from the evaluator to choose

Fold	Accuracy	Precision	Recall	MCC†
1	0.781	0.765	0.807	0.563
2	0.783	0.774	0.797	0.565
3	0.795	0.777	0.821	0.592
4	0.784	0.777	0.802	0.568
5	0.784	0.768	0.812	0.569
6	0.790	0.764	0.840	0.582
7	0.778	0.770	0.797	0.556
AVG	0.785	0.771	0.811	0.571

Table A.1: Results for BCeval in a 7-fold cross validation test on the dataset TRAINCH. † MCC = Matthews' Correlation Coefficient.

the best alignment among a set of alternatives. In particular, the correlation between the actual performance for a series of alignments and an evaluation metric for that alignment was analyzed. Alignments were obtained with the Needleman and Wunsch algorithm, scored with the BLOSUM62 matrix and gap opening/extension penalty of 2/12. The performance of an alignment was measured using the 'SSMA distance' (SSMAD), i.e. the average distance from a reference structural alignment A_{str}, obtained using SSAP(170) for each substitution in the sequence alignment A_{seq}:

$$\text{SSMAD} = \sum_{i}^{N} \left| A_{str_i} - A_{seq_i} \right| / N \tag{A.9}$$

where i indicates a substitution, N is the number of substitutions, A_i indicates the position of the substitution in alignment A. The mean of the evaluations for the target protein $\tilde{p}(bp_{tgt})$ has been taken as a metric. Figure A.6 plots the correlation between this metric and SSMAD on dataset TESTALIGN while Figure A.7 plots the correlation between the metric and RMSD of the models obtained for the same alignments with MODELLER(63; 64)[1] (used in fully automatic mode once supplied with the alignment). The Pearson's correlation coefficient is -0.386 for SSMAD and -0.520 for the RMSD.

[1]Only 637 models were obtained from the alignments owing to problems in the automatic process which extracted the indexes for the domains from the PDB files. The problem is often caused by domains which include non-consecutive parts of sequence.

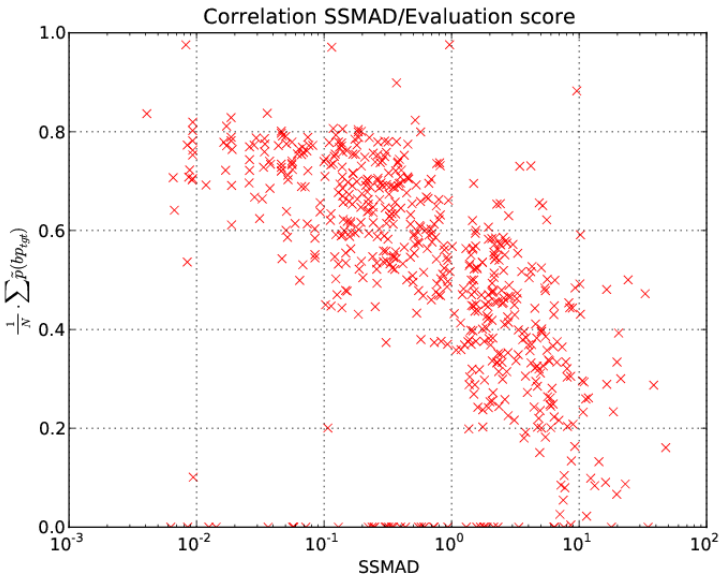


Figure A.6: BCeval score *vs.* SSMAD.

A.3.1.1 BCalign

A first set of experiments was run to find the best parameter set for the alignment algorithm. The set of parameters which gave the best results was $c_{gap_{op}} = 22$, $c_{gap_{ext}} = 9$ $c_{gap_{\beta}} = 6$, $\gamma_{abs} = 75$ and $\gamma_{rel} = 5$. The substitutions are scored using BLOSUM45. The algorithm appears to be better suited to distant homologues, since for sequence alignments between close homologues a standard sequence alignment is usually sufficiently reliable, and usually very few SSMAs are detected. If a different substitution matrix is used, all the parameters should be adjusted so as to obtain the best performance. The maximum number of iterations was set to 5. In order to restrict the time of the experiments, a limit of one minute was imposed on the search algorithm at each iteration. The experiments were run on a laptop with an Intel SU9600 CPU. The main code of the alignment algorithm was written in Java and experiments were scripted using Python via the Jython 2.5 interpreter[1], which supports calling Java routines natively.

On the TESTALIGN dataset, BCalign shows a relative improvement[2] of 6.2% (0.661 *vs.* 0.703 average) in the percent of correct substitutions against the same tech-

[1] `http://www.jython.org`.

[2] Relative improvements are calculated with $RI(a,b) = \frac{a-b}{(a+b)/2} \cdot 100$

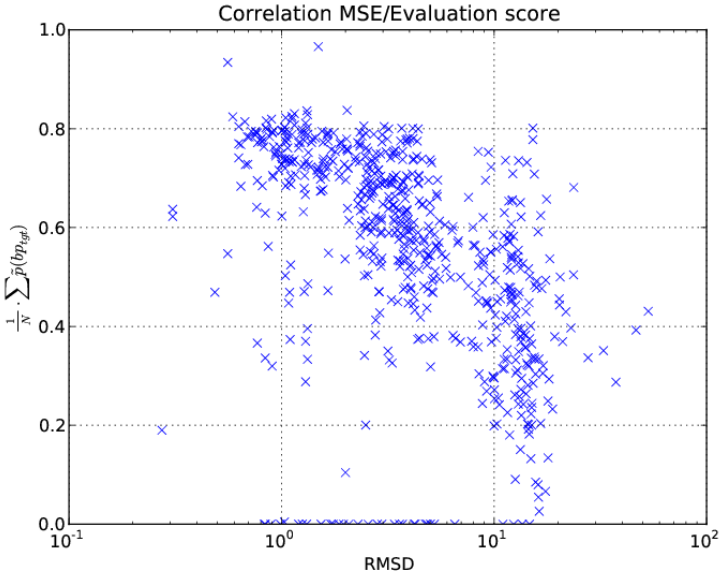


Figure A.7: BCeval score *vs.* RMSD (Å).

nique without the use of the evaluator ($\gamma_{abs} = 0$, $\gamma_{rel} = 0$). The latter configuration is referred to as NOBCalign. The SSMAD improvement is 15.6% (2.19 *vs.* 1.87 average). The average RMSD is 5.99Å for BCalign and 6.40Å for NOBCalign, an improvement of 6.59% as a result of using the BCeval predictor. The large values of RMSD result from the fact that the majority of the alignments in the test set have sequence identity below 25%. In addition, as seen in Figure A.7, a few models have extremely large RMSDs, greatly influencing the mean value.

In addition, BCalign has been compared with the multiple alignment technique, MUSCLE(29). MUSCLE was run with standard parameters, and all the homologous sequences in TESTDOM (according to the CATH definitions) were included in the multiple alignments. BCalign shows an average number of correct substitutions which is slightly better (+1.4%) than MUSCLE in the test; on the other hand, on average it does somewhat worse than MUSCLE in SSMAD and RMSD (−2.1% and −2.6%, respectively).

Better results are obtained by restricting comparisons to data for which we expect BCalign to perform well. Taking only the all-β and α/β domains (92% of the total in the test set, thus excluding the all-α domains where BCeval provides no useful information)

the RMSD improves by +6.78% with respect to NOBCalign. For the structures with at least 8 beta pairs (58% of the alignments) the RMSD improvement is 8.83% over NOBCalign, and 1.22% over MUSCLE.

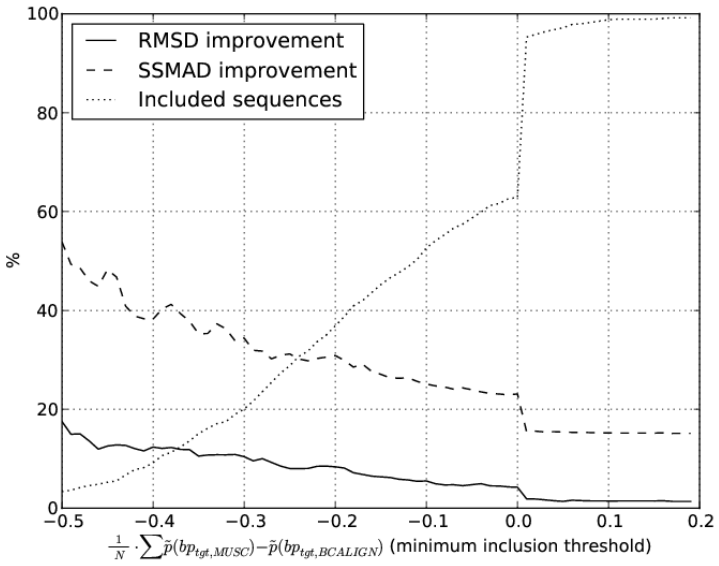


Figure A.8: Average improvement in SSMAD and RMSD with different inclusion thresholds. The threshold consists of the difference in the BCeval score between alignments obtained with BCalign and MUSCLE. The percent of included alignments at each threshold is also shown.

Further, the BCeval scores can be used to select those cases where BCeval makes confident predictions. Figure A.8 and A.9 show the average relative improvement in the RMSD and SSMAD between BCalign pairwise alignment and MUSCLE multiple alignment, when varying an inclusion threshold based on the improvement in the assignment of beta pairs when comparing MUSCLE and BCalign alignments, as evaluated using BCeval. The graphs clearly show that using an inclusion threshold of less than -0.3 (thus including up to 20% of alignments), substantial improvements in SSMAD and RMSD can be obtained.

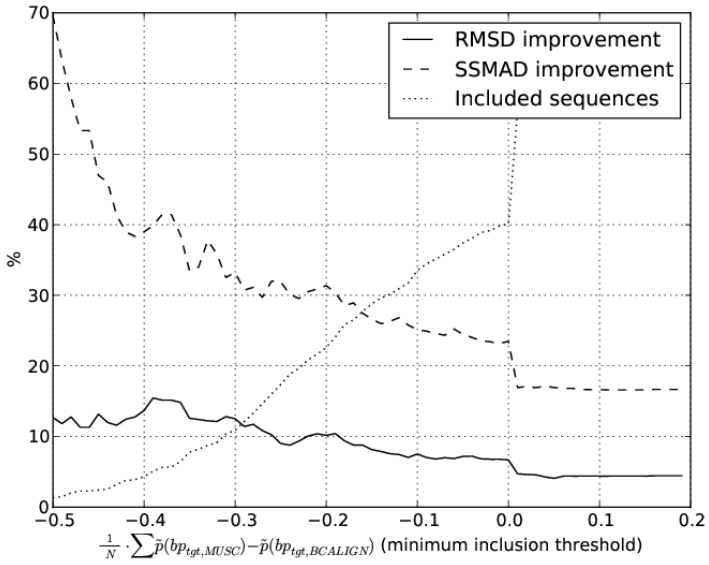


Figure A.9: Average improvement in SSMAD and RMSD with different inclusion thresholds, with number of beta pairs ≥ 8. The threshold consists of the difference in the BCeval score between alignments obtained with BCalign and MUSCLE. The percent of included alignments at each threshold is also reported.

A.4 Discussion

The prediction of protein tertiary structure has become one of the most important tasks in bioinformatics. Although building complete protein tertiary structures *de novo* is mostly not a tractable task, comparative modelling techniques are able to provide accurate models by assigning the structure from a template. Sequence alignment with the template protein is the most critical task in comparative modelling: a strong correlation holds between the RMS deviation of models and the occurrence of errors in the alignment.

In order to improve alignments in this context, we have exploited the likelihood of a given pairing between beta strands being correct. Since the location of beta strands is known for the template it can be assigned to the target sequence after the alignment. We have presented a beta contact evaluator, BCeval, which estimates the likelihood of assigned beta pairings occurring in real proteins by using a mixture of neural networks.

BCeval is then actively exploited with a novel sequence alignment technique, called

BCalign. We firstly proposed a scoring system, which modifies a normal system based on a substitution matrix by using the contribution of the evaluator. Since it is not possible to use standard dynamic programming with this scoring system, BCalign resorts to search algorithms, guided by an external loop to control the maximum run time.

Our experiments show a considerable correlation between the evaluations made by BCeval and the correctness of the assigned pairings. In particular, the quality of an alignment, measured by the average distance of substitutions from the correct position (SSMAD), is negatively correlated with a BCeval metric, consisting of the sum of the evaluations for each pair assigned from the template structure. This fact suggests that BCeval is indeed useful for choosing the best alignment from a set of alternatives obtained with different techniques and/or parameters.

Experiments confirm the validity of the approach: BCeval predictions show a considerable correlation with correct beta pair assignments and alignments obtained with BCalign show that the evaluation of assigned beta pairs can be successfully exploited to enhance pairwise sequence alignments.

Overall, BCalign showed a considerable improvement in pairwise alignments of 15.6% in SSMAD on a set of 743 alignments of domains not showing any homology with the data used to train the evaluator. Three-dimensional models obtained from the alignments with the same proteins show an average RMSD improvement of 6.6% compared with standard Needleman and Wunsch sequence alignments. Also, BCeval results are, on average, comparable with multiple alignments obtained with MUSCLE. However, choosing the 20% best scoring alignments according to the evaluator, models obtained with BCalign show a considerable improvement in the RMSD of about 10% over MUSCLE.

A.5 Conclusions

In conclusion, BCalign appears to perform best when used in a mixed environment, in which different techniques compete while taking into account the scores assigned by BCeval. Figure A.8 shows that restricting the use of BCalign to those cases where it makes the most confident predictions of an improvement in beta-strand pairing over MUSCLE, greatly increases its effectiveness. If only the best 20% of improved alignments (as predicted by BCeval) are used, the RMSD of the models was about 10%

better than multiple alignments generated using MUSCLE. Even including the best 50% of the alignments shows BCalign to be a good strategy (5% improvement over MUSCLE).

Finally, given that a number of novel techniques have been introduced together, the implementation of the algorithms can probably be further improved. The computation is still not sufficiently efficient, frequently reaching the time limit for long sequences. Of course longer sequences will, on average, have more beta-strands which can be exploited by the method and therefore are likely to show the best improvements. Clearly increasing the run-time limit or using faster machines may lead to further improvements, but more consistent improvements could be obtained by enhancing the search algorithm (in particular, a better heuristic function could greatly decrease the explored alternative paths). Another approach may be to devise a better control loop able to include the evaluations without overloading the search algorithm. Alternatively, stochastic local search algorithms, including genetic algorithms, could also be tried in order to have better control over the run time while using the same scoring systems.

Both BCeval and BCalign have been released as web servers and stand alone programs. Servers and executable can be accessed from `http://iasc.diee.unica.it/bcserver`.

References

[1] W. R. Taylor. **The Classification of Amino acid Conservation**. *J Theor Biol*, **119**:205–218, 1986. xiii, 13

[2] Cynthia Gibas and Per Jambeck. *Developing Bioinformatics Computer Skills*. O'Reilly Media, Inc., April 2001. xiii, 20

[3] G. Armano, F. Ledda, and E. Vargiu. **Sum-Linear Blosum: A Novel Protein Encoding Method for Secondary Structure Prediction**. *International Journal Communications of SIWN*, **6**:71–77, 2009. 3, 106, 126

[4] F. Ledda and E. Vargiu. **Experimenting Heterogeneous Output Combination to Improve Secondary Structure Predictions**. In *Workshop on Data Mining and Bioinformatics*, Cagliari (Italy), 2008. 3

[5] G. Armano, F. Ledda, and E. Vargiu. *SSP2: A Novel Software Architecture for Predicting Protein Secondary Structure*. Sequence and Genome Analysis: Methods and Application, in press. 3

[6] G. Armano and F. Ledda. **Exploiting Intra-Structure Information for Secondary Structure Prediction with Multifaceted Pipelines**. submitted September 2010. 3

[7] Filippo Ledda and Giuliano Armano andrew C.R. Martin. **Using a Beta Contact Predictor to Guide Pairwise Sequence Alignments for Comparative Modelling**. submitted October 2010. 3

[8] F. Ledda, L. Milanesi, and E. Vargiu. **GAME: A Generic Architecture based on Multiple Experts for Predicting Protein Structures**. *International Journal Communications of SIWN*, **3**:107–112, 2008. 3, 61, 150

[9] G. Armano, F. Ledda, and E. Vargiu. **GAME: a Generic Architecture based on Multiple Experts for bioinformatics applications**. In *BITS Annual Meeting 2009*, Genova (Italy), 2009. 3, 61

[10] W Kabsch and C Sander. **Dictionary of protein secondary structure: pattern recognition of hydrogen bonded and geometrical features**. *Biopolymers*, **22**:2577–2637, 1983. 15, 48

[11] D. Frishman and P. Argos. **Knowledge-based Protein Secondary Structure Assignment**. *Proteins*, **23**:566–579, 1995. 15

[12] Claus A. F. Andersen and Burkhard Rost. **Automatic secondary structure assignment**. *Methods Biochem Anal.*, **44**:341–363, 2003. 15

[13] C. B. Anfinsen. **Principles that Govern the Folding of Protein Chains**. *Science*, **181**:223–230, 1973. 16, 35, 49

[14] J. D. Levinthal and H. Rubin. **Serum Induced Changes in the fine Structure of Primary Chick Embryo Cultures**. *Exp Cell Res*, **52**:667–672, 1968. 17, 36

[15] M. Karplus and G. A. Petsko. **Molecular Dynamics Simulations in Biology**. *Nature*, **347**:631–639, 1990. 17

[16] S. T. Rao and M. G. Rossmann. **Comparison of super-secondary structures in proteins**. **76**(2):241–256, May 1973. 17

[17] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, B. A. Rapp, and D. L. Wheeler. **GenBank**. *Nuc. Ac. Res.*, **30**:17–20, 2002. 21

[18] H M Berman, J Westbrook, Z Feng, G Gilliland, T N Bhat, H Weissig, I N Shindyalov, and P E Bourne. **The Protein Data Bank**. *Nucleic Acids Research*, **28**:235–242, 2000. 22

[19] R. A. Sayle and E. J. Milner-White. **RASMOL: biomolecular graphics for all**. *Trends Biochem. Sci.*, **20**:374, 1995. 23

[20] **The PyMOL Molecular Graphics System**. 23

[21] C. A. Orengo, A. D. Michie, S. Jones, D. T. Jones, M. B. Swindells, and J. M. Thornton. **CATH–a Hierarchic Classification of Protein Domain Structures**. *Structure*, **5**:1093–1108, 1997. 23, 152

[22] A. G. Murzin, S. E. Brenner, T. Hubbard, and C. Chothia. **SCOP: a Structural Classification of Proteins Database for the Investigation of Sequences and Structures**. *J Mol Biol*, **247**:536–540, 1995. 23, 126

[23] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. **A model of evolutionary change in proteins**. In *Atlas of Protein Sequence and Structure*, pages 345–352, 1978. 25

[24] S. Henikoff and J. G. Henikoff. **Amino acid substitution matrices from protein blocks**. In *Proceedings of the National Academy of Sciences of the United States of America*, **89(22)**, pages 10915–10919, 1992. 26

[25] G. Vogt, T. Etzold, and P. Argos. **An Assessment of Amino acid Exchange Matrices in Aligning Protein Sequences: the Twilight zone Revisited**. *J Mol Biol*, **249**:816–831, 1995. 26

[26] S. B. Needleman and C. D. Wunsch. **A general method applicable to the search for similarities in the amino acid sequence of two proteins**. *J. Mol. Biol.*, **48**:443–453, 1970. 26, 149

[27] T. F. Smith and M. S. Waterman. **Identification of common molecular subsequences**. *J. Mol. Biol.*, **147**:195–197, 1981. 26

[28] D Higgins, J Thompson, T Gibson, J D Thompson, D G Higgins, and T J Gibson. **CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice**. *Nucleic Acids Research*, **22**:4673–4680, 1994. 27, 106

[29] Robert C. Edgar. **MUSCLE: a Multiple Sequence Alignment Method with Reduced time and Space Complexity**. *BMC Bioinformatics*, **5**:113–113, 2004. 27, 163

[30] C. Notredame, D.G. Higgins, and J. Heringa. **T-coffee: a novel method for fast and accurate multiple sequence alignment1**. *Journal of molecular biology*, **302**(1):205–217, 2000. 27

[31] J.G. Henikoff and S. Henikoff. **Using substitution probabilities to improve position-specific scoring matrices**. *Bioinformatics*, **12**(2):135, 1996. 28

[32] W. R. Pearson and D. J. Lipman. **Improved tools for biological sequence comparison**. *Proc. Natl. Acad. Sci. USA*, **85**:2444–2448, 1988. 29

[33] S F Altschul, W Gish, W Miller, E W Myers, and D J Lipman. **Basic local alignment search tool**. *Journal of Mol. Biology*, **215(3)**:403–410, 1990. 29

[34] S F Altschul, T L Madden, A A Schaeffer, J Zhang, Z Zhang, W Miller, and D J Lipman. **Gapped BLAST and PSI-BLAST: a new generation of protein database search programs**. *Nucleic Acids Research*, **25**:3389–3402, 1997. 29, 51

[35] K. Karplus, C. Barrett, and R. Hughey. **Hidden markov models for detecting remote protein homologies**. *Bioinformatics*, **14**(10):846–856, 1998. 29, 44, 106

[36] Abdi H., Valentin D., and Edelman B. *Neural networks*. Thousand Oaks (CA): Sage, 1999. 31

[37] Bishop C. M. *Neural networks for pattern recognition*. Oxford University Press, 1995. 31

[38] Duda R., Hart P.E., and Stork D.G. *Pattern classification*. Wiley, 2001. 31, 60

[39] G. Cybenko. **Approximation by superpositions of a sigmoidal function**. *Mathematics of Control, Signals, and Systems (MCSS)*, **2**(4):303–314, 1989. 32

[40] J.J. Hopfield. **Neural networks and physical systems with emergent collective computational abilities**. *Proceedings of the National Academy of Sciences of the United States of America*, **79**(8):2554, 1982. 32

[41] F.J. Pineda. **Generalization of back-propagation to recurrent neural networks**. *Physical Review Letters*, **59**(19):2229–2232, 1987. 32

[42] T. Kohonen. **The self-organizing map**. *Proceedings of the IEEE*, **78**(9):1464–1480, 2002. 32

[43] D.E. Rumelhart, G. E. Hinton, and R. J. Williams. **Parallel Distributed Processing, Explorations in the Microstructure of Cognition**. **1**:318–362, 1986. 33

[44] C.A. Floudas, H.K. Fung, S.R. McAllister, M. Monnigmann, and R. Rajgaria. **Advances in Protein Structure Prediction and De Novo Protein Design: A Review**. *Chemical Engineering Science*, **61**:966–988, 2006. 36

[45] Tom L. Blundell, Devon Carney, Stephen Gardner, Fiona Hayes, Brendan Howlin, Tim Hubbard, John Overington, Diljeet Athwal Singh, B. Lynn Sibanda, and Michael J. Sutcliffe. **Knowledge-based Protein Modelling and Design**. *Eur. J. Biochem.*, **172**:513–520, 1988. 36

[46] L. E. Donate, S. D. Rufino, L. H. Canard, and T. L. Blundell. **Conformational analysis and clustering of short and medium size loops connecting regular secondary structures: a database for modeling and prediction.** *Protein Sci.*, **5**:2600–2616, 1996. 37

[47] D. F. Burke and C. M. Deane. **Improved protein loop prediction from sequence alone**. *Protein Eng.*, **14**:473–478, 2001. 37

[48] R. E. Bruccoleri and M. Karplus. **Prediction of the Folding of Short Polypeptide Segments by Uniform Conformational Sampling**. *Biopolymers*, **26**:137–168, 1987. 37

[49] M. Feig, P. Rotkiewicz, A. Kolinski, J. Skolnick, and C. L. Brooks. **Accurate Reconstruction of All-atom Protein Representations from Side-chain-based Low-resolution Models**. *Proteins: Struct., Funct., Genet.*, **41**:86–97, 2000. 37

[50] J. W. Ponder and F. M. Richards. **Tertiary Templates for Proteins. Use of Packing Criteria in the Enumeration of Allowed Sequences for Different Structural Classes**. *J. Mol. Biol.*, **193**:775–791, 1987. 37

[51] S. Liang and N. V. Grishin. **Side-chain modeling with an optimized scoring function**. *Protein Sci.*, **11**:322–331, 2001. 37

[52] R. L. Dunbrack and M. Karplus. **Backbone-dependent Rotamer Library for Proteins: Application to Side-chain prediction.** *J. Mol. Biol.*, **230**:543–574, 1993. 37

[53] M. J. Bower, F. E. Cohen, and Jr Dunbrack, R. L. **Prediction of protein side-chain rotamers from a backbone-dependent rotamer library: a new homology modeling tool**. *J. Mol. Biol.*, **267**:1268–1282, 1997. 37

[54] Adrian A. Canutescu, Andrew A. Shelenkov, and Roland L. Dunbrack. **A Graph-theory Algorithm for Rapid Protein Side-chain Prediction**. *Protein Sci.*, **12**:2001–2014, 2003. 37

[55] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus. **A program for marcromolecular energy, minimization, and dynamics calculations**. *J. Comp. Chem.*, **4**:187–217, 1983. 37

[56] L. Kale, R. Skeel, M. Bhandarkar, R. Brunner, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadarajan, and K. Schulten. **NAMD2: Greater scalability for parallel molecular dynamics.** *J. Comput. Phys.*, **151**:283–312, 1999. 37

[57] M. J. Sippl. **Recognition of errors in three-dimensional structures of proteins.** *Proteins: Struct., Funct., Genet.*, **17**:355–362, 1993. 37

[58] R. Sánchez and A. Šali. **Evaluation of Comparative Protein Structure Modeling by MODELLER-3**. *Proteins: Struct., Funct., Genet.*, **1**:50–58, 1997. 37

[59] M. J. Sutcliffe, I. Haneef, D. Carney, and T. L. Blundell. **Knowledge based modelling of homologous proteins. 1. Three-dimensional frameworks derived from simultaneous superposition of multiple structures.** *Protein Eng.*, **1**:377–384, 1987. 37

[60] M. J. Sutcliffe, F. R. F. Hayes, and T. L. Blundell. **Knowledge based modelling of homologous proteins. 2. Rules for the conformations of sustituted side chains**. *Protein Eng.*, **1**:385–392, 1987. 37

[61] M. C. Peitsch. **ProMod and Swiss-Model: Internet-based tools for automated comparative modelling**. *Biochem. Soc. Trans. (London)*, **24**:274–279, 1996. 37

[62] K. Arnold, L. Bordoli, J. Kopp, and T. Schwede. **The SWISS-MODEL Workspace: A web-based environment for protein structure homology modelling**. *Bioinformatics*, **22**:195–201, 2006. 37

[63] A. Šali and T. L. Blundell. **Comparative protein modelling by satisfaction of spatial restraints**. *J. Mol. Biol.*, **234**:779–815, 1993. 37, 161

[64] A. Fiser, R. K. Do, and A. Šali. **Modeling of loops in protein structures**. *Protein Sci.*, **9**:1753–1773, 2000. 37, 161

[65] P. A. Bates and M. J. Sternberg. **Model building by comparison at CASP3: using expert knowledge and computer automation**. *Proteins: Struct., Funct., Genet.*, **37**:47–54, 1999. 37

[66] K. Ogata and H. Umeyama. **An automatic homology modeling method consisting of database searches and simulated annealing**. *J. Mol. Graph.*, **18**:305–306, 2000. 37

[67] Christophe Lambert, Nadia Léonard, Xavier De Bolle, and Eric Depiereux. **ESyPred3D: Prediction of Proteins 3D Structures**. *Bioinformatics*, **18**:1250–1256, 2002. 37

[68] Mark A. DePristo, Paul I. W. De Bakker, Reshma P. Shetty, and Tom L. Blundell. **Discrete Restraint-based Protein Modeling and the Calpha-trace Problem**. *Protein Sci*, **12**:2032–2046, 2003. 37

[69] A. C. R. Martin, M. W. MacArthur, and J. M. Thornton. **Assessment of comparative modeling in CASP2**. *Proteins: Struct., Funct., Genet.*, **Suppl. 1**:14–28, 1997. 37, 148

[70] S. Cristobal, A. Zemla, D. Fischer, L. Rychlewski, and A. Elofsson. **A Study of Quality Measures for Protein Threading Models**. *Bioinformatics*, **2**:5–5, 2001. 37

[71] C. Chothia. **Proteins. One thousand families for the molecular biologist.** *Nature*, **357**(6379):543, 1992. 37, 51, 135

[72] E.V. Koonin, Y.I. Wolf, and G.P. Karev. **The structure of the protein universe and genome evolution**. *Nature*, **420**(6912):218–223, 2002. 37

[73] L. Rychlewski, W. Li, L. Jaroszewski, and A. Godzik. **Comparison of sequence profiles. Strategies for structural predictions using sequence information**. *Protein Science*, **9**(2):232–241, 2000. 38

[74] G. Yona and M. Levitt. **Within the twilight zone: a sensitive profile-profile comparison tool based on information theory**. *Journal of Molecular Biology*, **315**(5):1257–1275, 2002. 38

[75] J.U. Bowie, R. Luthy, and D. Eisenberg. **A method to identify protein sequences that fold into a known three-dimensional structure**. *Science(Washington)*, **253**(5016):164–164, 1991. 38

[76] DT Jones, WR Taylor, and JM Thornton. **A new approach to protein fold recognition.** *Nature*, **358**(6381):86–89, 1992. 38

[77] R.B. Russell, R.R. Copley, and G.J. Barton. **Protein fold recognition by mapping predicted secondary structures**. *Journal of molecular biology*, **259**(3):349–365, 1996. 38

[78] B Rost. **Protein fold recognition by prediction-based threading**. *Journal of Mol. Biology*, **270**:1–10, 1997. 38

[79] K. Karplus, K. Sj
"olander, C. Barrett, M. Cline, D. Haussler, R. Hughey, L. Holm, and C. Sander. **Predicting protein structure using hidden Markov models**. *Proteins: Structure, Function, and Bioinformatics*, **29**(S1):134–139, 1997. 38

[80] Bystroff C. and Shao Y. **Fully automated ab initio protein structure prediction using I-SITES, HMMSTR and ROSETTA**. *Bioinformatics*, **18 Suppl 1**:S54–61, 2002. 38

[81] Liam J. McGuffin and David T. Jones. **Improvement of the GenTHREADER Method for Genomic Fold Recognition**. *Bioinformatics*, **19**(7):874–881, 2003. 38

[82] An Suei Yang and Lu Yong Wang. **Local Structure Prediction with Local Structure-based Sequence Profiles**. *Bioinformatics*, **19**(10):1267–1274, 2003. 38

[83] Klepeis J.L., Floudas C.A., Morikis D., Tsokos C.G., Argyropoulos E., Spruce L., and Lambris J.D. **Integrated computational and experimental approach for lead optimization and design of compstatin variants with improved activity**. *Journal of the American Chemical Society*, **125**:8422–8423, 2003. 38

[84] Przybylski D. and Rost B. **Improving fold recognition without folds**. *Journal of Mol. Biology*, **341**:255–269, 2004. 38

[85] Ren-Xiang Yan, Jing-Na Si, Chuan Wang, and Ziding Zhang. **DescFold: a web Server for Protein fold Recognition**. *BMC Bioinformatics*, **10**:416–416, 2009. 38

[86] P. Bradley, D. Chivian, J. Meiler, K. Misura, C.A. Rohl, W.R. Schief, W.J. Wedemeyer, O. Schueler-Furman, P. Murphy, J. Schonbrun, et al. **Rosetta predictions in CASP5: successes, failures, and prospects for complete automation**. *Proteins: Structure, Function, and Bioinformatics*, **53**(S6):457–468, 2003. 38

[87] B. Rost and C. Sander. **Conservation and prediction of solvent accessibility in protein families**. *Proteins: Structure, Function, and Bioinformatics*, **20**(3):216–226, 1994. 39

[88] G. Pollastri, P. Baldi, P. Fariselli, and R. Casadio. **Prediction of coordination number and relative solvent accessibility in proteins**. *Proteins: Structure, Function, and Bioinformatics*, **47**(2):142–153, 2002. 39

[89] D.T. Jones and J.J. Ward. **Prediction of disordered regions in proteins from position specific score matrices**. *Proteins: Structure, Function, and Bioinformatics*, **53**(S6):573–578, 2003. 39

[90] Jianlin Cheng and Pierre Baldi. **Improved Residue Contact Prediction Using Support Vector Machines and a Large Feature set**. *BMC Bioinformatics*, **8**:113–113, 2007. 39, 150

[91] Allison N. Tegge, Zheng Wang, Jesse Eickholt, and Jianlin Cheng. **NNcon: Improved Protein Contact map Prediction Using 2D-recursive Neural Networks**. *Nucleic Acids Res*, **37**:W515–W518, 2009. 39, 150

[92] Benjamin R. Jefferys, Lawrence A. Kelley, and Michael J. E. Sternberg. **Protein Folding Requires Crowd Control in a Simulated cell**. *J Mol Biol*, **397**:1329–1338, 2010. 39

[93] JY Paquet, C. Vinals, J. Wouters, JJ Letesson, and E. Depiereux. **Topology prediction of Brucella abortus Omp2b and Omp2a porins after critical assessment of transmembrane beta strands prediction by several secondary structure prediction methods.** *Journal of biomolecular structure & dynamics*, **17**(4):747, 2000. 41

[94] K. De Fays, A. Tibor, C. Lambert, C. Vinals, P. Denoel, X. De Bolle, J. Wouters, J.J. Letesson, and E. Depiereux. **Structure and function prediction of the Brucella abortus P39 protein by comparative modeling with marginal sequence similarities**. *Protein engineering*, **12**(3):217, 1999. 41

[95] *Using Secondary Structure Information to Perform Multiple Alignment*, 2004. 41

[96] Perutz M. F., Rossmann M. G., Cullis A. F., Will G. Muirhead G., and North A. T. **Structure of haemoglobin: A three-dimensional Fourier synthesis at 5.5 resolution, obtained by X-ray analysis**. *Nature*, **185**:416422, 1960. 43

[97] P. Y. Chou and U. D. Fasman. **Prediction of protein conformation**. *Biochemistry*, **13**:211–215, 1974. 43

[98] J. Garnier, DJ Osguthorpe, and B. Robson. **Analysis of the accuracy and implications of simple methods for predicting the secondary structure of globular proteins**. *Journal of Molecular Biology*, **120**(1):97–120, 1978. 43

[99] J. Garnier, J.F. Gibrat, and B. Robson. **GOR method for predicting protein secondary structure from amino acid sequence.** *Methods in enzymology*, **266**:540, 1996. 43

[100] B Robson. **Conformational Properties of Amino Acid Residues in Globular Proteins**. *Journal of Mol. Biology*, **107**:327–356, 1976. 44

[101] E M Mitchell, P J Artymiuk, D W Rice, and P Willett. **Use of Techniques Derived from Graph Theory to Compare Secondary Structure Motifs in Proteins**. *Journal of Mol. Biology*, **212**:151–166, 1992. 44

[102] M Kanehisa. **A Multivariate Analysis Method for Discriminating Protein Secondary Structural Segments**. *Protein Engineering*, **2**:87–92, 1988. 44

[103] T.M. Yi and E.S. Lander. **Protein Secondary Structure Prediction Using Nearest-neighbor Methods**. *J. Mol. Biol*, **232**:1117–1129, 1993. 44

[104] N Qian and T J Sejnowski. **Predicting the secondary structure of globular proteins using neural network models**. *Journal of Mol. Biology*, **202**:865–884, 1988. 44

[105] H. Bohr, J. Bohrand S. Brunak, R.M.J. Cotterill, and B. Lautrup et al. **Protein secondary structure and homology by neural networks**. *FEBS Letters*, **241**:223–228, 1988. 44

[106] H L Holley and M Karplus. **Protein Secondary Structure Prediction with a Neural Network**. *Proc. Natl. Acad. Sc.*, **86**:152–156, 1989. 44

[107] O B Ptitsyn and A V Finkelstein. **Theory of Protein Secondary Structure and Algorithm of its Prediction**. *Biopolymers*, **22**:15–25, 1983. 44

[108] W R Taylor and J M Thornton. **Prediction of Super-Secondary Structure in Proteins**. *Journal of Mol. Biology*, **301**:540–542, 1983. 44

[109] B. Rost and C. Sander. **Improved prediction of protein secondary structure by use of sequence profiles and neural networks**. *Proc Natl Acad Sci U S A*, **90**(16):7558–7562, 1993. 44, 45, 106, 120

[110] J. M. Chandonia and M. Karplus. **Neural networks for secondary structure and structural class prediction**. *Protein Science*, **4**:275–285, 1995. 44, 45, 120

[111] S K Riis and A Krogh. **Improving prediction of protein secondary structure using structured neural networks and multiple sequence alignments**. *Journal of Comp. Biology*, **3**:163–183, 1996. 44

[112] A. A. Salamov and V. V. Solovyev. **Protein secondary structure prediction using local alignments**. *Journal of Mol. Biology*, **268**:31–36, 1997. 44

[113] D T Jones. **Protein secondary structure prediction based on position-specific scoring matrices**. *Journal of Mol. Biology*, **292**:192–202, 1999. 44, 45, 56, 106

[114] J. A. Cuff and G. J. Barton. **Application of multiple sequence alignment profiles to improve protein secondary structure prediction.** *Proteins*, **40**(3):502–511, 2000. 44, 99, 106

[115] P Baldi, S Brunak, P Frasconi, G Soda, and G Pollastri. **Exploiting the Past and the Future in Protein Secondary Structure Prediction**. *Bioinformatics*, **15**:937–946, 1999. 44, 45, 57, 106

[116] G. Pollastri and A. McLysaght. **Porter: a new, accurate server for protein secondary structure prediction**. *Bioinformatics*, **21**(8):1719–20, 2005. 44, 45, 58

[117] V. V. Solovyev and A. A. Salamov. **Predicting a-helix and b-strand segments of globular proteins**. *CABIOS*, **10**:661–669, 1994. 44

[118] R.D. King and M.J. Sternberg. **Identification and application of the concepts important for accurate and reliable protein secondary structure prediction**. *Protein Sci.*, **5**:2298–2310, 1996. 44

[119] A. A. Salamov and V. V. Solovyev. **Prediction of Protein Secondary Structure by Combining Nearest-Neighbor Algorithms and Multiple Sequence Alignment**. *Journal of Mol. Biology*, **247**:11–15, 1995. 44, 45

[120] D. Frishman and P. Argos. **75% accuracy in protein secondary structure prediction**. *Proteins*, **27**:329–335, 1997. 44, 45

[121] C. Bystroff, V. Thorsson, and D. Baker. **HMMSTR: A hidden markov model for local sequence-structure correlations in proteins**. *Journal of Mol. Biology*, **301**:173–190, 2000. 44, 45

[122] S. Hua and Z. Sun. **A novel method of protein secondary structure prediction with high segment overlap measure: support vector machine approach**. *Journal of molecular biology*, **308**(2):397–407, 2001. 44

[123] J. J. Ward, L. J. McGuffin, B. F. Buxton, and D. T. Jones. **Secondary structure prediction with support vector machines.** *Bioinformatics*, **19**(13):1650–1655, 2003. 44, 106

[124] J. A. Cuff and G. J. Barton. **Evaluation and improvement of multiple sequence methods for protein secondary structure prediction.** *Proteins*, **34**(4):508–19, 1999. 44

[125] Giuliano ARMANO, G. MANCOSU, L. MILANESI, A. ORRO, M. SABA, and E. VARGIU. **A Hybrid Genetic-Neural System for Predicting Protein Secondary Structure**. *BMC BIOINFORMATICS*, **6 (suppl. 4):s3**, 2005. 44, 109

[126] M Ouali and R King. **Cascaded multiple classifiers for secondary structure prediction**. *Protein Science*, **9**:1162–1176, 1999. 44, 45

[127] H. Lin, J. Chang, K. Wu, T. Sung, and W. Hsu. **HYPROSP II-A knowledge-based hybrid method for protein secondary structure prediction based on local prediction confidence**. *Bioinformatics*, **21**(15):3227–3233, 2005. 44, 112, 122

[128] Xin-Qiu Yao, Huaiqiu Zhu, and Zhen-Su She. **A Dynamic Bayesian Network Approach to Protein Secondary Structure Prediction**. *BMC Bioinformatics*, **9**:49–49, 2008. 44, 126, 136

[129] Cole C, Barber JD, and Barton GJ. **The Jpred 3 secondary structure prediction server**. *Nucleic Acids*, **36**(2):W197–W201, 2008. 45

[130] J. Moult, R. Judson, K. Fidelis, and J.T. Pedersen. **A large-scale experiment to assess protein structure prediction methods**. *Proteins*, **23(3)**:iiiv, 1995. 45

[131] A. V. Eyrich, M. A. Mart-Renom, D. Przybylski, M. S. Madhusudhan, A. Fiser, F. Pazos, A. Valencia, A. Sali, and B. Rost. **EVA: continuous automatic evaluation of protein structure prediction servers**. *Bioinformatics*, **17(12)**:1242–1243, 2001. 45, 126

[132] A Zemla, C Vencolvas, K Fidelis K, and B Rost. **A modified definition of SOV, a segment-based measure for protein secondary structure prediction assesment**. *Proteins*, **34**:220–223, 1999. 46

[133] B.W. Matthews. **Comparison of the predicted and observed secondary structure of T4 phage lysozyme**. *Biochim. Biophys. Acta*, **405**:442–451, 1975. 46, 160

[134] B. Rost. **Review: Protein Secondary Structure Prediction Continues to Rise**. *Journal of Struct. Biology*, **134**:204–218, 2001. 48

[135] J. M. Chandonia and M. Karplus. **The importance of larger data sets for protein secondary structure prediction with neural networks**. *Protein Science*, **5**:768–774, 1996. 50

[136] Gianluca Pollastri, Alberto Martin, Catherine Mooney, and Alessandro Vullo. **Accurate prediction of protein secondary structure and solvent accessibility by consensus combiners of sequence and structure information**. *BMC Bioinformatics*, **8**(1):201, 2007. 52, 58

[137] G E Crooks and S E Brenner. **Protein secondary structure: entropy, correlations and prediction**. *Bioinformatics*, **20**:1603–1611, 2004. 53

[138] R. Schneider C. Sander. **Database of homology-derived protein structures and the structural meaning of sequence alignment**. *Proteins*, **9**(1):56–68, 1991. 55, 106

[139] C. Cortes and V. Vapnik. **Support-vector networks**. *Machine learning*, **20**(3):273–297, 1995. 60

[140] L Breiman. **Bagging Predictors**. *Machine Learning*, **24**(2):123–140, 1996. 60, 83

[141] R E Schapire. **A Brief Introduction to Boosting**. In *Proc. of the 16th Int. Joint Conference on Artificial Intelligence – IJCAI 99*, pages 1401–1406, 1999. 60, 83

[142] Erin L. Allwein, Robert E. Schapire, and Yoram Singer. **Reducing multiclass to binary: a unifying approach for margin classifiers**. *J. Mach. Learn. Res.*, **1**:113–141, 2001. 60, 83, 103

[143] A. Sharkey, N. Sharkey, U. Gerecke, and G. Chandroth. **The test and select approach to ensemble combination**. *Multiple Classifier Systems*, pages 30–44, 2000. 60

[144] G. Giacinto and F. Roli. **An approach to the automatic design of multiple classifier systems**. *Pattern Recognition Letters*, **22**(1):25–33, 2001. 60

[145] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. **The WEKA Data Mining Software: An Update**. *SIGKDD Explorations*, **11**(1), 2009. 60

[146] Ingo Mierswa, Michael Wurst, Ralf Klinkenberg, Martin Scholz, and Timm Euler. **YALE: Rapid Prototyping for Complex Data Mining Tasks**. In Lyle Ungar, Mark Craven, Dimitrios Gunopulos, and Tina Eliassi-Rad, editors, *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 935–940, New York, NY, USA, August 2006. ACM. 60

[147] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989. 65

[148] J.H. Holland. *Progress in Theoretical Biology*, chapter Adaption, pages 4:263–293. Rosen, R., Snell, F.M. (eds.), New York Academic Press, 1976. 65

[149] S.W. Wilson. **Genetic Algorithms in Search, Optimization and Machine Learning**. *Evolutionary Computation*, **3(2)**:149–175, 1995. 65

[150] R.A. Jacobs, M.I. Jordan, S.J. Nowlan, and G.E. Hinton. **Adaptive Mixtures of Local Experts**. *Neural Computation*, **3**:79–87, 1991. 65

[151] A.S. Weigend, M. Mangeas, and A.N. Srivastava. **Nonlinear Gated Experts for Time Series: Discovering Regimes and Avoiding Overfitting**. *Int. Journal of Neural Systems*, **6**:373–399, 1995. 65

[152] Giuliano ARMANO. *Applications of Learning Classifier Systems*, chapter NXCS Experts for Financial Time Series Forecasting, pages 68–91. Springer Verlag, 2004. 65

[153] M. Fowler. *Refactoring*. Addison-Wesley, 1999. 67

[154] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995. 67

[155] David H. Wolpert. **Stacked Generalization**. *Neural Networks*, **5**:241–259, 1992. 83

[156] J.J. Rodriguez, L.I. Kuncheva, and C.J. Alonso. **Rotation Forest: A New Classifier Ensemble Method**. *PAMI*, **28**(10):1619–1630, October 2006. 83

[157] I. T. Jolliffe. *Principal Component Analysis*. Springer Series in Statistics, 2002. 84

[158] Giuliano ARMANO, G. MANCOSU, A. ORRO, and E. VARGIU. **A Multi-Agent System for Protein Secondary Structure Prediction**. *LECTURE NOTES IN COMPUTER SCIENCE*, **3**:14–32, 2005. Transactions on Computational Systems Biology III, Lecture Notes in Computer Science, C. Priami, E. Merelli, P. Gonzalez, A. Omicini (eds). 94

[159] Thornton JM Jones DT, Taylor WR. **A model recognition approach to the prediction of all-helical membrane protein structure and topology**. *Biochemistry*, **33**:3038–3049, 1994. 97

[160] T.G. Dietterich and G. Bakiri. **Solving multiclass learning problems via error-correcting output codes**. *Arxiv preprint cs/9501101*, 1995. 103

[161] R. E. Dickerson, R. Timkovich, and R. J. Almassy. **The cytochrome fold and the evolution of bacterial energy metabolism**. *Journal of Mol. Biology*, **100**:473–491, 1976. 106

[162] M. J. Zvelebil, G. J. Barton, W. R. Taylor, and M. J. E. Sternberg. **The cytochrome fold and the evolution of bacterial energy metabolism**. *Journal of Mol. Biology*, **195**:957–961, 1987. 106

[163] B. Rost. **PHD: Predicting one-dimensional protein structure by profile based neural networks.** *Methods Enzymol*, **266**:525–539, 1996. 106

[164] L. R. Rabiner and B. H. Juang. **An introduction to hidden Markov models**. *IEEE ASSP Magazine*, pages 4–15, January 1986. 106

[165] J. Platt, Ô.
"OÓ, and Ó.Ø. ÓÑ. **Probabilistic outputs for support vector machines**. *Bartlett P. Schoelkopf B. Schurmans D. Smola, AJ, editor, Advances in Large Margin Classifiers*, pages 61–74. 119

[166] T.N. Petersen, C. Lundegaard, M. Nielsen, H. Bohr, J. Bohr, S. Brunak, G.P. Gippert, and O. Lund. **Prediction of protein secondary structure at 80% accuracy**. *Proteins*, **41**:17–20, 2000. 120

[167] G. Wang and R.L. Dunbrack. **PISCES: a protein sequence culling server**. *Bioinformatics*, **19**(12):1589, 2003. 143

[168] P.E. Hart, N.J. Nilsson, and B. Raphael. **A Formal Basis for the Heuristic Determination of Minimum Cost Paths**. *Systems Science and Cybernetics, IEEE Transactions on*, **4**(2):100 –107, jul. 1968. 143, 159

[169] I. N. Shindyalov and P. E. Bourne. **Protein Structure Alignment by Incremental Combinatorial Extension (CE) of the Optimal path**. *Protein Eng.*, **11**:739–747, 1998. 147

[170] W. R. Taylor and C. A. Orengo. **Protein Structure Alignment**. *J. Mol. Biol.*, **208**:1–22, 1989. 147, 161

[171] S. Subbiah, D. V. Laurents, and M. Levitt. **Structural Similarity of DNA-binding Domains of Bacteriophage Repressors and the Globin core**. *Curr. Biol.*, **3**:141–148, 1993. 147

[172] L. Holm and C. Sander. **Protein Structure Comparison by Alignment of Distance Matrices**. *J. Mol. Biol.*, **233**:123–138, 1993. 147

[173] Takeshi Kawabata. **MATRAS: A Program for Protein 3D Structure Comparison**. *Nuc. Ac. Res.*, **31**:3367–3369, 2003. 147

[174] J. F. Gibrat, T. Madej, and S. H. Bryant. **Surprising Similarities in Structure Comparison**. *Curr. Opin. Struct. Biol.*, **266**:540–553, 1996. 147

[175] E. Krissinel and K. Henrick. **Secondary-structure Matching (SSM), a new tool for fast Protein Structure Alignment in Three Dimensions**. *Acta Crystallogr.*, **60**:2256–2268, 2004. 147

[176] M. Novotny, D. Madsen, and G. J. Kleywegt. **Evaluation of Protein fold Comparison Servers**. *Proteins: Struct., Funct., Genet.*, **54**:260–270, 2004. 147

[177] M. A. S. Saqi, R. B. Russell, and M. J. E. Sternberg. **Misleading local sequence alignments: implications for comparative modelling**. *Protein Eng.*, **11**:627–630, 1998. 148

[178] C. A. Orengo, A. D. Michie, S. Jones, D. T. Jones, M. B. Swindells, and J. M. Thornton. **CATH–a Hierarchic Classification of Protein Domain Structures**. *Structure*, **5**:1093–1108, 1997. 149

[179] Frances Pearl, Annabel Todd, Ian Sillitoe, Mark Dibley, Oliver Redfern, Tony Lewis, Christopher Bennett, Russell Marsden, Alistair Grant, David Lee, Adrian Akpor, Michael Maibaum, Andrew Harrison, Timothy Dallman, Gabrielle Reeves, Ilhem Diboun, Sarah Addou, Stefano Lise, Caroline Johnston, Antonio Sillero, Janet Thornton, and Christine Orengo. **The CATH Domain Structure Database and Related Resources Gene3D and DHS Provide Comprehensive Domain Family Information for Genome Analysis**. *Nuc. Ac. Res.*, **33**:D247–D251, 2005. 149

[180] H. M. Fooks, A. C. R. Martin, D. N. Woolfson, R. B. Sessions, and E. G. Hutchinson. **Amino acid Pairing Preferences in Parallel Beta-sheets in Proteins**. *J Mol Biol*, **356**:32–44, 2006. 149, 152

[181] S. Lifson and C. Sander. **Specific Recognition in the Tertiary Structure of Beta-sheets of Proteins**. *J Mol Biol*, **139**:627–639, 1980. 149

[182] M. A. Wouters and P. M. Curmi. **An Analysis of side Chain Interactions and pair Correlations Within Antiparallel Beta-sheets: the Differences Between Backbone Hydrogen-bonded and Non-hydrogen-bonded Residue Pairs**. *Proteins*, **22**:119–131, 1995. 149

[183] E. G. Hutchinson, R. B. Sessions, J. M. Thornton, and D. N. Woolfson. **Determinants of Strand Register in Antiparallel Beta-sheets of Proteins**. *Protein Sci*, **7**:2287–2300, 1998. 149, 152

[184] Marco Lippi and Paolo Frasconi. **Prediction of protein ?-residue contacts by Markov logic networks with grounding-specific weights**. *Bioinformatics*, **25**(18):2326–2333, 2009. 151

[185] W. Kabsch and C. Sander. **Dictionary of Protein Secondary Structure: Pattern Recognition of Hydrogen-bonded and Geometrical Features**. *Biopolymers*, **22**:2577–2637, 1983. 153

[186] R. F. Smith and T. F. Smith. **Pattern-induced multi-sequence alignment (PIMA) algorithm employing secondary structure-dependent gap penalties for use in comparative protein modelling**. *Protein Eng.*, **5**:35–41, 1992. 155

[187] R.E. Korf. **Depth-First iterative-deepening: An optimal admissible tree search**. *Artificial Intelligence*, **27**(1):97–109, 1985. 160

List of Publications Related to the Thesis

Published papers

Journal papers

- F. Ledda, L. Milanesi, and E. Vargiu. **GAME: A Generic Architecture based on Multiple Experts for Predicting Protein Structures**. *International Journal Communications of SIWN*, **3**:107–112, 2008. Relation to Chapter 4.
- G. Armano, F. Ledda, and E. Vargiu. **Sum-Linear Blosum: A Novel Protein Encoding Method for Secondary Structure Prediction**. *International Journal Communications of SIWN*, **6**:71–77, 2009. Relation to Chapter 5.

Book Chapters

- G. Armano, F. Ledda, and E. Vargiu. *SSP2: A Novel Software Architecture for Predicting Protein Secondary Structure*. Sequence and Genome Analysis: Methods and Application, in press. Relation to Chapter 6.

Conference papers

- G. Armano, F. Ledda, and E. Vargiu. **GAME: a Generic Architecture based on Multiple Experts for bioinformatics applications**. In *BITS Annual Meeting 2009*, Genova (Italy), 2009. Relation to Chapter 4.
- F. Ledda and E. Vargiu. **Experimenting Heterogeneous Output Combination to Improve Secondary Structure Predictions**. In *Workshop on Data Mining and Bioinformatics*, Cagliari (Italy), 2008. Relation to Chapter 6 .

Submitted papers

- G. Armano and F. Ledda. **Exploiting Intra-Structure Information for Sec-**

ondary Structure Prediction with Multifaceted Pipelines. submitted September 2010. Relation to Chapter 6.

- Filippo Ledda, Giuliano Armano, and Andrew C.R. Martin. **Using a Beta Contact Predictor to Guide Pairwise Sequence Alignments for Comparative Modelling**. submitted October 2010. Relation to Appendix A.

Declaration

I herewith declare that I have produced this paper without the prohibited assistance of third parties and without making use of aids other than those specified; notions taken over directly or indirectly from other sources have been identified as such. This paper has not previously been presented in identical or similar form to any examination board.

The thesis work was conducted from January 2007 to December 2010; this document was completed in January 2011.

Cagliari, February 16th, 2011

Filippo Ledda