NTNU
Innovation and Creativity

Protein Remote Homology Detection using Motifs made with Genetic Programming

Tony Håndstad

Master of Science in Computer Science
Submission date: July 2006
Supervisor: Arne Halaas, IDI
Co-supervisor: Pål Sætrom, Interagon

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

Genetic programming for remote homology prediction

Homology prediction is the problem of deciding whether two protein sequences have evolved from a common ancestor protein. Predicting homology becomes increasingly difficult as sequence similarities decrease and predicting remote homologies remains one of the most challenging problems in computational biology.

The best and most recent methods for predicting remote homology are discriminative and often use support vector machines to train classifiers that recognize sequences of common evolutionary origin. One such method uses occurrences of pregenerated motifs to measure similarities between sequences. The method, called eMOTIF, gives promising results, but its performance might be improved by using motifs that specifically characterize the sequences to be classified. This project will test whether using genetic programming to create specialized motifs improves the performance of motif-based similarity measures in remote homology prediction.

Assignment given: 10. February 2006
Supervisor: Arne Halaas, IDI

Abstract

A central problem in computational biology is the classification of related proteins into functional and structural classes based on their amino acid sequences. Several methods exist to detect related sequences when the level of sequence similarity is high, but for very low levels of sequence similarity the problem remains an unsolved challenge. Most recent methods use a discriminative approach and train support vector machines to distinguish related sequences from unrelated sequences. One successful approach is to base a kernel function for a support vector machine on shared occurrences of discrete sequence motifs. Still, many protein sequences fail to be classified correctly for a lack of a suitable set of motifs for these sequences.

We introduce a motif kernel based on discrete sequence motifs where the motifs are synthesised using genetic programming. The motifs are evolved to discriminate between different families of evolutionary origin. The motif matches in the sequence data sets are then used to compute kernels for support vector machine classifiers that are trained to discriminate between related and unrelated sequences.

When tested on two updated benchmarks, the method yields significantly better results compared to several other proven methods of remote homology detection. The superiority of the kernel is especially visible on the problem of classifying sequences to the correct fold. A rich set of motifs made specifically for each SCOP superfamily makes it possible to classify more sequences correctly than with previous motif-based methods.

Preface

The picture on the front page of this thesis illustrates the three dimensional structure of a protein. These molecules, whose form could easily be taken for pieces of modern art, play crucial roles in all biological systems. A central element in the understanding of cells is to understand the structure and function of these key building blocks of nature. In the recent years, breakthrough developments in large-scale sequencing have led to a surge of available protein sequence information, but this leap in sequence information has not been matched by the number of protein structures available in the Protein Data Bank. It is therefore a great need for new automatic methods that can cluster proteins into categories of structure and functionality.

This thesis presents the results of our work on the problem of protein remote homology detection. Our main contribution is a method that uses discrete sequence motifs generated using genetic programming as a basis for an SVM kernel. The genetic programming process is accelerated by special-purpose hardware from Interagon AS. The thesis is divided into six chapters. After the introduction in chapter one, a theoretical basis is given in chapter two. Chapter three explains the ideas behind our contributions. Chapter four presents the results and chapter five gives an additional discussion on some of the findings. Chapter six is a conclusion. An article based on our main findings are currently in the review process for BMC Bioinformatics. The article is included in appendix C.

Many thanks to supervisor Pål Sætrom for his interest and outstanding guidance with this thesis.

TRONDHEIM, 4. JULY, 2006

TONY HÅNDSTAD

Contents

List of Figures

Chapter 1

Introduction

The sciences of biology is nowadays witnessing a major paradigm shift. New techniques in molecular biology are helping researchers transform biology from a qualitative and descriptive science to a quantitative and predictive engineering science. These new high-throughput techniques are also making molecular biology a special information-rich science. The analysis and handling of this information, being insurmountable without computers, has spawned the field of bioinformatics whose ultimate goal is to aid the researchers in viewing molecular biology in a bottom-up manner as complex systems whose properties can be simulated and predicted in silico.

The information overload currently experienced is especially evident in the very high quantities of sequence information being made available at a rapid rate through the public databases. This is widening an already existing gap between the number of protein sequences and the much smaller number of proteins whose structure and function has been experimentally determined. Researchers are therefore relying on automatic methods to classify new protein sequences into functional and structural families. This is feasible because even though there exists an enormous amount of different proteins, most of them share some of the smaller number of structural motifs or folds, partly as a result of evolution conserving certain successful structures.

So because protein structure is more conserved in evolution than sequence is, proteins that have diverged from a common ancestor often share structural and functional similarities while retaining only small amounts of sequence similarity. An important problem in bioinformatics is the problem of detecting these subtle sequence similarities as this might imply that the sequences are homologues and thus possibly share structural and functional similarities as well.

Much progress has been made on detecting subtle sequence similarities since

the first solutions that compared a new sequence to another by alignment appeared in the 1970s. Because the sequence similarities between to homologue sequences often are very small, it becomes challenging to separate these similarities from the possible false positives, that is, sequences that share small similar regions by chance. More recent efforts have thus looked for better opportunities to distinguish homologues from other sequences. Discriminative methods, such as the very successful support vector machine learning algorithm, have proven to give the best results.

The support vector machine learning algorithm is trained on both homologue and non-homologue sequences by using only a similarity measure between the sequences. The similarity measure comes in form of kernel functions and most recent research has been looking for better methods to compare sequences and produce more discriminative kernels. Alignment scores, string patterns content, and motif content are some of the types that have been explored.

Protein sequence motifs are conserved regions in related sequences with a high degree of sequence similarity and often represent active domains in the proteins. One way to build a kernel for a support vector machine classifier is to base it on the motif content of two sequences and simply count the number of common motifs. A problem with this approach is the availability of motifs. Even though there exist many motif databases, most of these cannot be used to test a motif based kernel because of the fact that the motifs are made from the sequences that would have participated in the test, and those who can be used may not have a sufficient suitable set of motifs to classify all sequences correctly.

We therefore propose a method to automatically synthesise discrete sequence motifs by looking for sequence similarities in the training sets of the classifier. The motifs are produced with genetic programming and rated based on their ability to describe similarities in the positive training set and not match other sequences. We then check all the sequences for their motif content and compute a kernel from the resulting feature vectors. We compare our method to several other standard methods in homologue detection on two benchmarks and find that our method performs significantly better.

This thesis is divided into six chapters. Chapter one is this introduction. Chapter two gives a background on the theory behind the main methods used in the thesis and describes earlier work on the problem of remote homology detection and automatic motif synthesis. Chapter three tells how experiments are conducted and explains what methods are used to validate the results. The results themselves are presented in chapter four. Chapter four discusses some of the details behind the results, but a more conclusive

discussion is found in chapter five. Chapter six gives the conclusion reached in this thesis.

Chapter 2

Background

This chapter gives a theoretic basis needed for understanding the rest of the thesis. The first section explains the nature of proteins. The following sections explain techniques in the field of machine learning with emphasis on genetic programming, boosting and kernel methods. A quick overview of the special-purpose hardware used is given in section 2.7. The last sections of the chapter are devoted to a small guide to sequence alignment, earlier work on automatic motif synthesis by genetic programming, and earlier work on remote homology detection.

2.1 Proteins

2.1.1 Proteins are amino acid polymers

Proteins are the most versatile macromulecules in living systems and serve crucial functions in essentially all biological processes [1]. This diversity in function is a result of the vast space of potential proteins available. As many other biological macromolecules, proteins are constructed from a limited number of building blocks.

Proteins are built from a repertoire of 20 different amino acids. These molecules have a common core and differ in only one part called the side chain, which gives the amino acids their specific chemical properties. These properties are crucial for how the amino acids react with other molecules, including water. Through a polymerization reaction, amino acids may create a chemical bond between the core parts and form a chain of amino acids, a polypeptide. A protein is such a long chain, folded into a 3-dimensional structure due to the different amino acid's reactions with themselves and the surrounding water molecules.

A protein's biological function is determined by its 3-dimensional structure. This structure is again in principle only determined by the sequence of the amino acids in the chain [2], often referred to as the protein's primary structure. Because of interactions between the core parts of close amino acids, nearby parts of the chain will form regularly repeating substructures such as alpha helixes and pleated sheats. The placement and relationships of these along the amino acid chain constitute the protein's secondary structure. The tertiary structure refers to the structure of the fully folded protein as a result of all interactions between amino acids and their environment. Further, if the protein also consists of several sub-units of amino acid chains, the quaternary structure refers to the spatial arrangement of the sub-units and the nature of their interactions. Figure 2.1 gives an overview of the different levels of protein structure.

Because the function of a protein depends on its structure, an important issue in biology is determining the structure of proteins. The two most common forms of determining structure experimentally is X-ray crystallography and NMR spectroscopy, but since these methods are very resource demanding, alternatives for determining tertiary structure from primary structure are necessary. Automatically determining protein structure from the amino acid sequence has only become more important after the breakthrough developments in high throughput sequencing in the last years. Though much research has been done, inferring a protein structure ab inito by calculating the folding process using physics is difficult and also very costly computationally. A more common approach is to compare a new sequence to a database of existing structures in a process known as threading, or to compare the sequence to an existing related sequence whose structure is already known. The last method is known as homology detection.

2.1.2 Sequence similarity imply homology

Two proteins are said to be homologous if they have evolved from the same common ancestor. Homologue proteins often have similar structure, but because there is less pressure to maintain sequence than structure similarity, homologue sequences are often only vaguely similar. Detecting this sequence similarity and clustering new sequences into homologue groups of structural likeness is therefore a challenging and important problem. Like many problems in bioinformatics, it is impossible to create a definite traditional algorithm that easily solves the problem of homology detection. Instead, what is sought is to let the computer learn to recognise what constitute homology, and thus modern techniques for protein homology detection are based on machine learning algorithms.

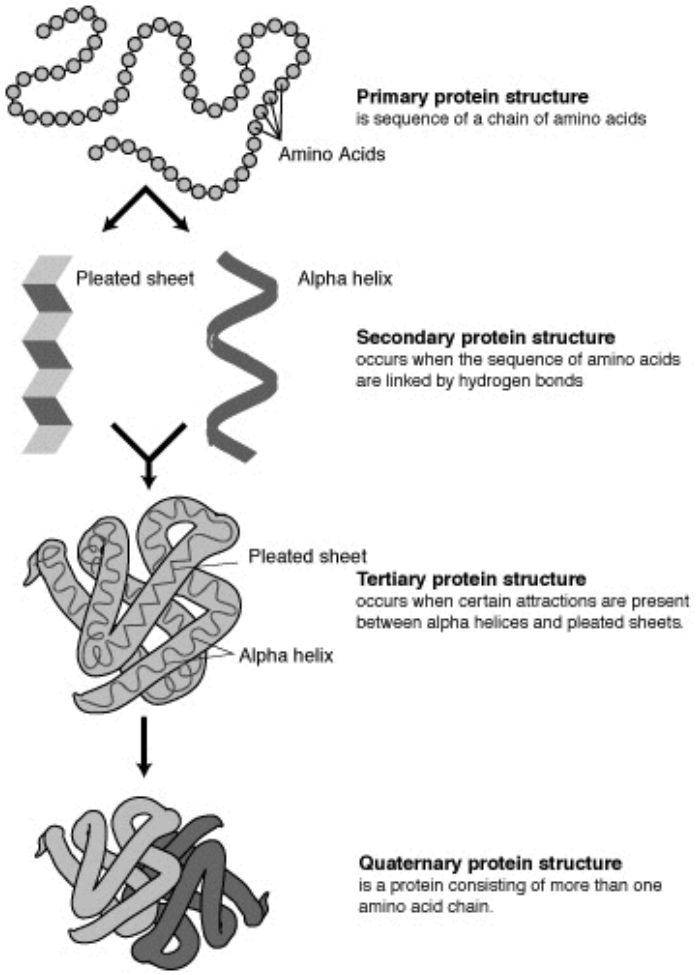

Figure 2.1: The different levels of protein structure. (Figure courtesy of NIH [3])

Often, a set of related proteins will be more similar in some areas and less in others. A protein sequence motif is a part of an amino acid sequence which is conserved across many different proteins. The motif is often an important part of a protein, such as an active site which cannot be modified without loss of function. Due to evolutionary pressure to maintain the motif, it will be conserved to a higher degree than the other parts of the proteins. The fact that homologue sequences often share sequence motifs is something we take advantage of in this thesis, as we use machine learning algorithms to classify sequences as homologues or non-homologues based on their sequence motif content.

2.2 Machine learning

2.2.1 Supervised learning for classification

Machine learning can be described as the study of computer algorithms that improve automatically through experience [4]. The two most important techniques used in this thesis are Boosted Genetic Programming and Support Vector Machines. Both of these are examples of supervised learning algorithms.

When using supervised learning for classification, the learning system is during training given pairs of patterns and labels. That is, the training set consists of a set of n pairs $\{(x_1, y_1), \ldots, (x_i, y_i), \ldots, (x_n, y_n)\}$ where x_i is a pattern and y_i is the corresponding label. If y can only take on two values, we have a binary pattern recognition problem. The goal for the learning system is to approximate a function f modelling the unknown probability distribution $P(x, y)$ so that the function is able to generalise and classify new patterns with the correct y value.

In other words, the machine learning algorithm is presented with a number of training examples and will from these learn to predict correct outputs to new inputs that may not have been encountered previously. Because the learning algorithm does not fully know the distribution $P(x, y)$ for all possible inputs x, it has to minimise the risk of predicting wrong labels by selecting the function that minimises errors on the training data. It is essential though, to limit the complexity of the function or else the problem of overfitting might occur.

2.2.2 Overfitting reduces performance of classifiers

A problem that may occur in machine learning is the phenomenon of overfitting. This implies that the learning system optimises on classifying the training set correctly, thereby sacrificing its generalisation power. The result is a more complex function tuned to the peculiarities of the training set and worse performance on unseen data. Overfitting can be countered by restricting the class of functions a learning machine can implement. How this is done, depends on the learning paradigm and the learning machine. If the risk of overfitting depends on the amount of training as in neural networks, one way to avoid it is to use validation techniques like cross validation and early stopping [5]. In the simple Figure 2.2, one of the alternative functions has overfitted to the training set, giving an error on unseen data.

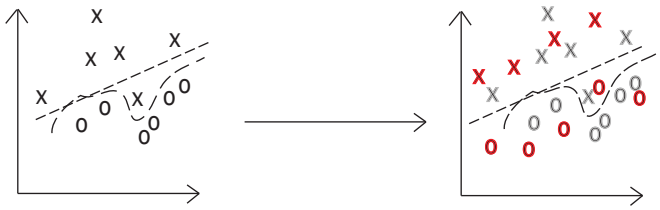

Figure 2.2: The more complex function has overfitted to the training set.

2.2.3 Validation methods for verifying performance

It is important to have a good procedure for verifying the quality of a classifier after training. In the simplest one, the dataset is split in two parts. One part is for training the classifier and one part for is for testing. It is important that the two parts are disjunct. A common error is to optimise the learning system's performance on the test set, which will give a wrong estimate of the performance of the classifier. If the learning algorithm requires additional parameters as input, an evaluation set may be used to optimise the parameter values. This is illustrated in Figure 2.3.

Another form of validation is K-fold cross validation. Here, a set of m examples are partioned into K sets (folds) of size m/K. For each fold, a classifier is trained on the other folds combined and then tested on the fold. The

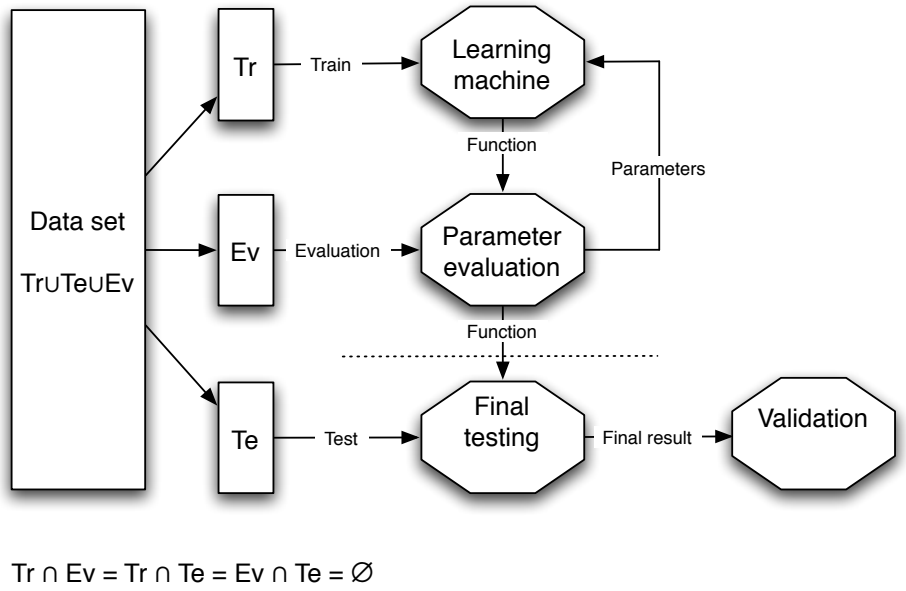

Figure 2.3: The different stages of the machine learning process have distinct data sets.

average error over all K partions is the cross-validated error rate. In a more computationally expensive variant called leave-one out validation, the value of K is equal to m.

2.3 Boosting improves accuracy

Boosting [6] is a way of improving the accuracy of any given learning algorithm. The idea is to combine the weighted results of several less complex classifiers (called weak learners) into one prediction for better accuracy. The weight of a weak learner is proportional to how accurate it is. To make the weak learners, the learning algorithm is repeatedly given the training set data with different weights on every example. One way to set the weights of the examples is to give highest weight to the training examples most often misclassified by previous runs. This is the approach taken by the algorithm AdaBoost [7], given in Algorithm 1.

Algorithm 1 The AdaBoost algorithm.

Input: $S = \{(x_1, y_1), ..., (x_N, y_N)\}$, Iterations T

Initialize: $d_n^{(1)} = 1/N$ for all $n = 1, ..., N$
for t =1 to T **do**
  Train classifier with respect to the weighted sample set {S,$d^{(t)}$} and obtain hypothesis $h_t : x \mapsto \{-1, +1\}$, i.e. $h_t = L(S, d^{(t)})$
  Calculate the weighted training error ϵ_t of h_t: $\epsilon_t = \sum_{n=1}^{N} d_n^{(t)}$ **I**$(y_n \neq h_t(x_n))$
  Set: $\alpha_t = \frac{1}{2} log \frac{1-\epsilon_t}{\epsilon_t}$
  Update weights: $d_n^{(t+1)} = d_n^{(t)} exp(-\alpha_t y_n h_t(x_n))/Z_t$ where Z_t is a normalisation constant, such that $\sum_{n=1}^{N} d_n^{(t+1)} = 1$
  if $\epsilon_t = 0$ or $\epsilon_t \geq \frac{1}{2}$ **then**
    Break and set $T = t - 1$.
  end if
end for
Output: $f_t(x) = \sum_{t=1}^{T} \frac{\alpha_t}{\sum_{r=1}^{T} \alpha_r} h_t(x)$

2.4 Genetic programming

2.4.1 GP is a variant of evolutionary algorithms

A genetic programming algorithm [8] is a variant of evolutionary algorithms, metaheuristic optimisation algoritms that use mechanisms inspired by natural evolution to do a parallell search using a population of candidate solutions. In genetic programming, the candidate solutions are variable length computer programs or data that may be interpreted as computer programs. These are traditionally represented as parse trees. A simple description of a genetic programming algorithm is given in Algorithm 2.

Algorithm 2 A basic genetic programming algorithm.

Initialise population with random candidate solutions
Evaluate each candidate
repeat
  Select parents
  Recombine pairs of parents
  Mutate the resulting offspring
  Evaluate new candidates
  Select individuals for the next generation
until Termination criteria satisfied

The evolution normally starts with a randomly generated population. In

each iteration of the algorithm (referred to as a generation), each candidate solution is evaluated by a fitness function. The candidate solutions with the highest fitness have a higher probability of being selected to continue to the next iteration.

The following operators work on the candidate solutions in every iteration in order to move the search forward:

- **Mutation** The mutation operator brings novelty into the population by randomly altering a solution. This helps the search avoid getting stuck in local maxima.
- **Crossover** This operator, illustrated in Figure 2.4, makes new solutions by combining parts of two or more solutions.
- **Selection** The selection operator selects which of the candidate solutions that will be the basis for the next generation. Several variants exists, and all work by giving a higher selection probability to candidate solutions with a better fitness. In this way, the selection operator pushes the search in the direction of the search space where good solutions have been seen previously.

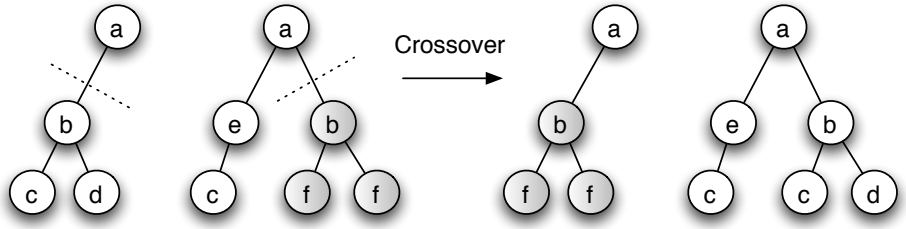

Figure 2.4: The crossover genetic operator.

2.4.2 GP is faster at finding optimal solutions than pure random search

Genetic programming is related to an earlier technique, genetic algorithms [9], where the candidate solutions are represented as fixed length bit strings encoding a domain specific phenotype. The Schema theorem of Holland [9] gives a theoretic analysis of how genetic algorithms differ from a parallell random search. Holland argues that each candidate solution in the population participates in numerous ways in the search process by containing schemata (partial building blocks of a good solution) and exposing these to

the evaluation function. As the search progresses, good schemata will tend to multiply in the population and the crossover operator will combine these good partial solutions into better solutions. This will speed up the search process. The building block hypothesis [8] is a similar, but more complex hypothesis for genetic programming.

2.5 Kernel methods

Many linear classifiers exist for classifying data, but they are unable to handle non linearly separable data. One solution for this problem is to map the data into a richer mathematical space where the data become linearly separable, and use a linear classifier in this space. Kernel methods [10] are a class of machine learning methods that uses mathematical kernel functions to implicitly map the data, thereby avoiding computational problems related to working in high dimensional spaces.

A kernel function is a function that returns the value of the dot product of two arguments projected into the space induced by the kernel function. The dot product might be seen as a measurement of similarity between the arguments. Kernel methods work by comparing all data instances with each other, thereby producing a matrix of pairwise comparisons which is the basis for several learning algorithms that can be written in terms of dot products. The matrix made from a kernel function must be positive definitive and symmetrical. As long as these restrictions are followed, the mathematical rewriting known as the "kernel trick" makes it possible to implicitly calculate in the more complex space where data are linearly separable. This enable non linearity for linear algorithms, as illustrated in Figure 2.5. All kernel methods build on these same principles, so the making of a kernel function and using the kernel output are two independent steps. This modularism means that using kernel methods for classification consists in choosing a specific kernel function and choosing a kernel based classifier.

Several standard kernel functions exist, from the simplest linear kernel to the more complex gaussian radial basis kernel. Of course, a problem specific kernel function is preferable compared to any general solution. Regardless of kernel used, the resulting matrix can be used with kernel methods such as the k-nearest neighbour classifer that classifies according to the majority of the class of the k nearest training instances in the feature space, or the more advanced support vector machine algorithm.

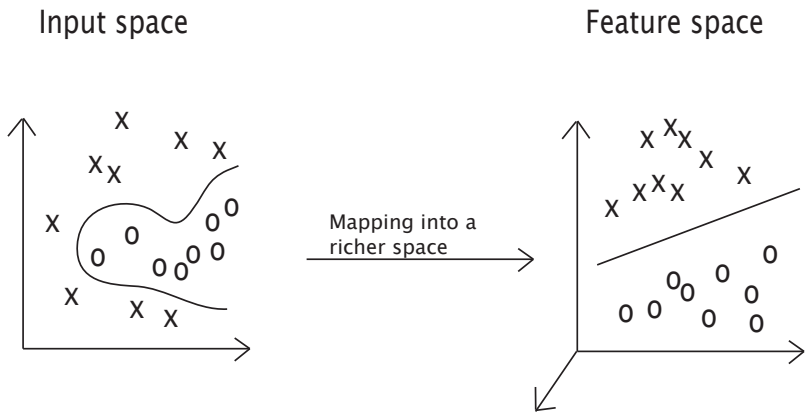

Figure 2.5: Kernel mapping makes data linearly separable in feature space.

2.6 Support Vector Machines

A support vector machine (SVM) is a linear learning machine that can be used for classification and regression [11]. In the simplest case of classification, the SVM tries to build as a decision function a linear hyperplane in the input space. In general there are several such possible hyperplanes to choose from. The SVM will select a number of data examples that lie close to the hyperplane, called support vectors. The SVM then chooses the hyperplane which lies in equal distance between the two classes of selected data, giving the maximal margin between the support vectors (Figure 2.6). This is motivated by the results of statistical learning theory [12] which give a minimised probabilistic generalisation error bound when the margin is maximised.

The real power of Support vector machines comes into play when they are used in combination with kernels. This makes the SVM a non linear classifier by replacing dot products in the algorithm with kernel evaluations. By choosing the maximum margin hyperplane, the SVM also minimises the risk of overfitting in the kernel induced space.

In the case when the data are linearly nonseparable, because of noise or misclassified data in the training set, the SVM can handle this by introducing soft margins [13]. The SVM will then still try to maximise the hyperplane margin between the support vectors as much as possible, but accept wrongly classified instances with a penalty. A penalty constant is then needed as a parameter to the SVM algorithm, controlling the tradeoff between achieving a large margin and the number of misclassifications.

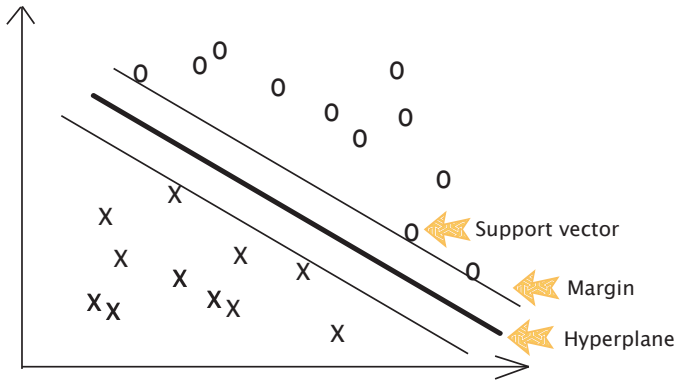

Figure 2.6: The margin between the hyperplane and the support vectors is maximised.

In every case, the decision function depends on a weighted sum of the support vectors and finding these weights is a quadratic optimisation problem. One solution to this problem is sequential minimal optimisation [11].

2.7 Hardware acceleration

The Interagon pattern matching chip [14] (PMC) is a special purpose VLSI architecture developed for rapid searching for multiple patterns in large datasets. The multiple instruction, single data stream architecture allows in theory up to 127 patterns to be searched simultaneously in up to 100MB of unstructured data pr second, when running on a clock speed of 100MHz. The patterns are similar to regular expressions but with additional functionality such as proximity, adjacency and order conditions on subpatterns and the ability to allow approximate matching until a certain error threshold. Up to 16 PMCs, each with dedicated memory, can be mounted on a standard PCI card giving the opportunity to search 1600MB of data pr second.

Programming of the PMC is facilitated by a special purpose query language, developed by Interagon AS [15]. The Interagon Query Language (IQL) is a superset of regular expressions with extensions to facilitate queries with features like those described above. Table 2.1 gives an overview of the IQL operators used for this thesis.

Table 2.1: IQL operators used.

Operator	Use	Example
Atomic characters	Terminals	ABC matches ABC
$\{< q >: p >=$	Minimum matches	$\{AA : p >= 1\}$ matches AB and BA
.	Wildcard	A.C matches AAC, ABC etc.
|	Disjunction	ABC|AAC matches both ABC and AAC

2.8 Bioinformatics

Like we already have discussed in the introduction and in section 2.1, the field of molecular biology is producing information at an astounding rate. A big part of this information is in the form of genome and protein sequences and one of the most common operations is to compare one sequence with another.

2.8.1 Pairwise sequence alignment

In pairwise sequence alignment, one tries to find the optimal alignment of two sequences so that the sequences have identical or similar characters on as many positions as possible. The reason for doing an alignment is to discover how similar two sequences are and also where they are similar. A scoring function is used to evaluate alternative alignments. Gaps, corresponding to possible character insertions or deletions, are inserted in the sequences at a cost to the score. When aligning protein sequences, one takes into account that it is more probable for a mutated amino acid to be retained during evolution if it has properties similar to the original amino acid. Thus the mismatches produced are also often variably penalised according to a substitution matrix.

In a global alignment, the complete sequences are aligned. The Needleman and Wunsch algorithm [16] is an example of an algorithm that does a global alignment. In a local alignment, the algorithm tries to align the most similar regions of the sequences. The Smith-Waterman algorithm [17] is an example of a local alignment algorithm. Despite that both of these algorithms use dynamic programming to find the optimal alignment, their exhaustive approach is considered too slow to be used in for example database searches, so heuristical alternatives like BLAST [18] and FASTA [19] that trade some sensitivity for speed are used instead.

The BLAST algorithm operates in three stages. In the first, short subwords of the search sequence is looked up in a pre-computed index of the database. In the second stage, the alignment of the subwords are extended in both directions without considering gaps or insertions. In the final stage, the high scoring alignments of the second stage are aligned with gaps using a variant of the Smith-Waterman algorithm and then these alignments are finally sorted and reported according to statistical relevance.

2.8.2 Multiple sequence alignment finds patterns among several sequences

The idea of aligning two sequences can be extended to alignment of multiple sequences. Because the problem of finding the optimal alignment of several sequences is NP-complete, the sequences are usually progressively aligned: Two and two sequences are aligned and then their alignment is aligned with that of two other sequences, building a tree structure where the root at the end is the alignment of all the sequences in the set. Clustal [20] is a popular algorithm that uses the progressive approach.

After alignment, the regions with the best alignments are of special interest. These are the regions in related sequences that, because of their biological relevance, are most conserved by evolution. They constitute a common pattern, or sequence motif, which can be represented in several ways.

The most simple model is the consensus sequence that for each position stores the most common character in the alignment. A more powerful method is to store a motif as a form of regular expression where for example each position can have alternative characters that match. A more quantitative model is the position specific weight matrix, or profile, which gives the relative frequency of every amino acid at every position. Further sequences can then be aligned to the profile and all alignments will produce a continuous value describing the fit, instead of just the binary match or no-match that is the result with a regular expression.

The PSI-BLAST algorithm [21] uses profiles to do iterative BLAST searches. In each iteration, the sequences of profile alignments scoring below a threshold value are added to the profile to give a more general model that can find more distant homologues than ordinary BLAST.

2.9 Earlier work on genetic programming for biological motif discovery

Another method that has been used to find motifs in sequences is genetic programming. The first well known use of genetic programming in biological motif discovery was that of Koza and Andre [22]. They evolved motifs using a small subset of the Prosite [23] pattern format, allowing only for character sets matching one or more residues in addition to the exact matching of residues. Olson [24] used genetic algorithms to evolve more Prosite-like patterns from multiple aligned sequences. Hu [25] used genetic progamming on unaligned sequences to evolve Prosite motifs. His algorithm also had a local optimisation step which refined expression terms denoting gaps and used an expanded set of terminals with domain specific amino acid substitution groups. Ross [26] used genetic programming with stochastic regular expressions to generate motifs on unaligned sequences. Finally, Seehuus et al. [27] showed with a larger study that a linear based genome might be better than a tree based genome when using genetic programming on automatic motif synthesis.

It is hard to compare these methods as their setup and data sets are very different to one another. Still, many of the methods evolved motifs comparable to or even better than Prosite motifs, illustrating the potential of genetic programming in automatic motif discovery.

2.10 Earlier work on remote homology detection

2.10.1 A development in four stages

The development of methods for detecting protein sequence similarities can be broken into four stages [28]. The most early methods looked for similarities between single pairs of proteins. In 1970, the very first method was developed by Needleman and Wunsch [16]. They introduced a dynamic programming algorithm that performed a global alignment of two sequences, giving a similarity score for the optimal alignment. The later Smith-Waterman [17] algorithm is a variation of the algorithm by Needleman and Wunsch, performing an optimal local alignment. Both these algorithms are guaranteed to find the optimal global or local alignment, but they also have a quadratic running cost. Therefore, heuristic alternatives have been developed and BLAST [18] and FASTA [19] are well known examples of this type. Both BLAST and FASTA are local alignment algorithms.

In the second stage, further accuracy was obtained by using profiles [29] and hidden Markov models [30] for representing aggregate statistics taken from

a set of similar sequences, and using these models to compare the statistics to a new protein of interest. These two models were later used in the third phase in combination with information in large databases of unlabelled protein sequences to iteratively collect homologous sequences and incorporate statistics into a single model. But these methods, such as PSI-BLAST [21] and SAM-T98 [31], used only statistics based on sequences that were known to be evolutionary related. The fourth stage was introduced with the idea of using both related and unrelated sequences and letting a machine learning method learn the differences between the related and unrelated sequences. This idea represented a big leap forward for the field of homology detection.

2.10.2 Discriminative modelling improves accuracy

The fourth stage was introduced in 1999. In their award winning paper, Jaakkola et al. [32] showed that better accuracy could be obtained by modelling the difference between positive and negative examples, that is, related and unrelated sequences. Their method, the Fisher kernel, trained a hidden Markov model on positive examples and also extra sequences taken from unlabelled databases for each family of related proteins. These models were later used for computing fixed length gradient vectors on every protein, positive or negative, and the labelled gradient vectors were then used in combination with a Support Vector Machine to give a very good result compared to earlier methods.

After the introduction of the Fisher kernel by Jaakkola, several other kernels have been developed for protein remote homology detection. Logan et al. [33] introduced the idea of using protein motifs as features for an SVM. A different motif kernel was described in 2003 by Ben-Hur and Brutlag [34]. Using the eBLOCKS database [35] and the eMOTIF method [36] to create discrete sequence motifs, they calculate their kernel by representing each protein sequence as a sparse vector of the eMOTIF content and then taking the dot product between all such vectors.

Liao and Noble described in 2002 [28] a kernel generated from pairwise sequence comparison scores from the Smith-Waterman algorithm. Several string kernels have also been developed. The spectrum kernel [37] uses a similarity score based on how many k-length substrings, called k-mers, two sequences share. The kernel is efficiently computed using a trie data structure instead of explicitly representing the protein sequences as vectors of k-mers. The spectrum kernel was later generalised to allow each k-mer to contain a specific number of mismatches. The resulting mismatch kernel [38] is more adapted to modelling biological phenomena in protein sequences, such as mutation.

2.10.3 Performance comparison of early kernel methods

There are two primary benchmarks normally used for evaluating the recent kernel methods, based on the SCOP database (see section 3.1 for more information). One benchmark includes in the training set additional non-SCOP homologs identified via a hidden Markov model as introduced in Jaakkola et al. [32], the other benchmark uses only SCOP domains [28]. The SVM-Fisher method has good performance when used with the first benchmark, but performs less well on the second [28], presumably because the hidden Markov models are undertrained. The SVM-Pairwise algorithm of Liao and Noble thus performs better than SVM-Fisher with trainingsets made only of SCOP domains [28].

The eMOTIF kernel of Ben-Hur and Brutlag seems to outperform SVM-Pairwise on the benchmark not including additional homologs [34]. However, later results [39] show that on different datasets and with different parameters, the SVM-pairwise can achieve equal or even superior performance. Even though the spectrum kernel does not perform as well as SVM-Fisher [37], its mismatch variant is comparable to SVM-Fisher on the first benchmark which includes extra homologs and to SVM-pairwise on the second benchmark [40]. In summary, this means that of the methods already mentioned in this section, the eMOTIF kernel and the SVM-Pairwise is considered to give the best classification performance on remote homology detection. It is surprising though, that the general spectrum and mismatch string kernels show such good performance compared to more biological relevant kernels.

2.10.4 Newer approaches use more powerful and biological relevant sequence models

There is continuous development in the field of remote homology detection and several new approaches have been introduced since the first kernel methods appeared. Though most methods are still based on kernels used in combination with a support vector machine, other approaches are also being investigated. Plötz and Fink [41] use profile hidden Markov models with features that better capture the biochemical properties of the amino acid residues in a local context, giving semi-continuous models that outperform earlier models based on hidden Markov models. Hou et al. [42] also try a more biological intuitive solution to the problem by using local structure motifs based on the I-sites library of 262 sequence-structure correlation patterns. Their extended work [43] also takes into account the sequence

of the structure motifs by using a previously made hidden Markov model [44], thereby greatly improving performance. Both methods have rather elaborate methods for creating feature vectors which they use to train a support vector machine. The first I-sites method performs comparable to SVM-Pairwise but with lower computational cost, the second method using HMMs has significant better performance than the SVM-Pairwise and Fisher kernel.

Another method that is based on the mismatch kernel is the profile kernel of Kuang et al. [39]. Instead of letting the k-mers have a certain number of mismatches as in the mismatch kernel, a distinct profile is computed for every sequence using PSI-BLAST and the k-mer is said to be present in the sequence if it achieves a summed log odds profile score along the whole profile length that is lower than a given threshold. The result is significantly better than earlier methods such as the eMOTIF kernel of Ben-Hur and SVM-Pairwise.

Weston et al. [45] investigate a way to incorporate the large quantities of unlabelled sequence data available into a base kernel to improve classification accuracy. The mismatch kernel [38] is here chosen as a base kernel. Using the feature vectors of the mismatch kernel, a BLAST or PSI-BLAST E-value neighbourhood is computed for each sequence, the feature vectors of the sequence are averaged with the neighbourhood and a kernel computed from such averaged pairs of sequences. The method yields very good results, comparable with that of Kuang et al. [39]; it scales worse as the number of sequences rise, but is very flexible regarding the choice of a base kernel.

2.10.5 State of the art sees a return to alignment methods

The latest methods to enter the field of remote homology detection take a different approach for creating kernels. Instead of coding the sequences as a set of feature vectors and then using a standard kernel function such as the dot product on these vectors to compute the Gram matrix, a more direct comparison between the sequences is sought. The local alignment kernel of Saigo et al. [46] measures the similarity of two sequences directly by computing all optimal local alignment scores with gaps between all of their possible subsequences. To make valid Mercer kernels, the smallest negative eigenvalue is subtracted from the diagonal of the similarity measure matrices. The performance of the method is found to be better than all earlier kernels not based on the use of sequence profiles.

Rangwala and Karypis borrow several earlier ideas when they introduce two classes of kernels in their 2005 paper [47]. They generate profiles for every

sequence and use a direct profile-to-profile similarity measure to examine the similarity of two sequences. The first class of kernels combines the ungapped alignment of subsequences found in the two sequences and produces a similarity measure by combining the sums of the direct profile comparisons between these subsequences. The other class finds the optimal local alignment (with gaps) that optimises a scoring function taking into account the profile-to-profile scores and gap costs. All similarity measures are converted to valid Mercer kernels using the same approach as the local alignment kernel [46].

2.10.6 Conclusion

Remote homology detection has seen a great improvement since the first methods appeared in the 1980s. Currently, the state of the art is based on an optimised local alignment, combined with a direct profile-based scoring scheme that creates a very good similarity measure for a support vector machine. The superiority of this method can be witnessed in the fact that even when using substitution matrices instead of profiles, the method still yields better results than other recent methods when classifying unknown sequences to the superfamily level or the fold level of SCOP [47]. Still, the good performance comes at a cost. Rangwala and Karypis run the PSI-BLAST on the entire non-redundant NCBI database for a full five iterations, a parameter study is done for optimisation and the local alignment also requires considerable running time, something the authors do not mention in their article. It is therefore still ample opportunity to improve the methods of remote homology detection, both in computational and biological performance.

Chapter 3

Methods

Our main contribution is to test a kernel based on motifs generated with genetic programming and see how this method compares with related and other methods on the problem of detecting protein homologues. This chapter explains the approach taken for all the experiments and shows how the results are generated and validated.

3.1 Validation methods and data sources

To simulate the problem of remote homology detection, the SCOP database's [48] classification of protein sequences is used as the basis for two benchmarks. The SCOP database aims to classify all proteins whose structure is known in a hierarchy based on structural and evolutionary relatedness. The major levels in the hierarchy are:

- **Family** Proteins clustered in a family have clear evolutionary relationship, meaning that pairwise residue identities between proteins are 30% and greater.
- **Superfamily** Proteins in superfamiles show low degrees of sequence identities, but structural and functional features in the proteins gives them a probable common evolutionary origin.
- **Fold** Proteins have the same common fold if they have the same major secondary structures in the same arrangement and with the same topological connections. This does not necessarily mean they have the same evolutionary origin.

The first benchmark is the one originally introduced by Jaakkola et al. [32]. The objective is to learn to classify an unknown protein sequence to the correct SCOP superfamily. The benchmark is modified as in the article of Liao and Noble [28] so that additional homologue proteins are not included. The classification is used in the following way (see Figure 3.1): For each

family, the protein domains within the family constitute a positive test set. Protein domains from the other families in the same superfamily constitute a positive training set. The other superfamilies are negative trainingsets with one randomly chosen family in every superfamily as the negative test set.

The second benchmark follows that of Rangwala and Karypis [47]. The benchmark is similar to the first, except that we move one level up in the SCOP hierarchy. The objective is thus to classify a sequence to the correct fold. A superfamily in a fold is a positive test set. The other superfamilies within the same fold go in the positive training set. Superfamilies in other folds constitute the negative training set, except for one random superfamily in every fold that will be added to the negative test set. At the fold level, the sequence similarity is very low and the sequences do not necessarily have the same evolutionary origin. This makes it considerably harder to classify a sequence correctly.

The two benchmarks make two datasets. Both sets use data from SCOP version 1.67 as released in February 2005. The sequences are first filtered using the Astral database [49], keeping only sequences having less than 95% identity. They are then filtered according to the principle that there should be at least 10 sequences for training and 10 sequences for testing each classifier. The first dataset will therefore be built from all families in a superfamily having one family with more than 10 sequences and 10 more sequences in total in the other families of the superfamily. The second dataset used for fold detection is built from all superfamilies in a fold having one superfamily with more than 10 sequences and 10 more sequnces in total in the other superfamilies of the fold.

This means that for the first dataset, 102 classificators will be trained using a total of 4019 sequences from 397 families in 53 superfamilies. For the second dataset, 86 classificators will be trained from 34 distinct folds. Here, 3840 sequences in a total of 374 superfamilies make up the dataset. Of the 3840 sequences in the fold benchmark, 2076 do not participate in the superfamiliy benchmark. The 102 families and 86 superfamilies tested in our superfamily and fold benchmarks are almost twice the number of families and superfamilies used in previous benchmark studies. The datasets are generated with Perl scripts from the Astral filtered FASTA file. Table 3.1 gives an overview of the two datasets.

Table 3.1: Summary of the two datasets.

Set	Classificators	Folds	Superfamilies	Families	Sequences
SF	102	45	53	397	4019
Fold	86	34	374	784	3840

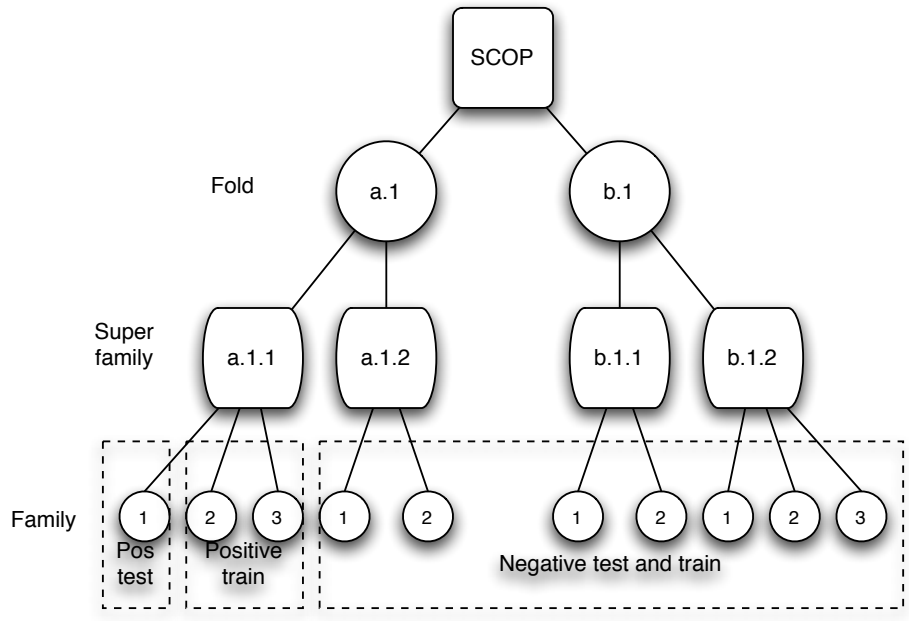

Figure 3.1: SCOP database superfamily benchmark

3.2 Methods explored

This section describes the different methods used to investigate remote homology detection. The section first introduces the eMOTIF kernel and shows how the idea of a motif kernel is extended further with additional motifs generated automatically. Then we look at how these methods compare against a boosted classifier, string and alignment based kernels and profile based methods.

3.2.1 The eMOTIF kernel

In their 2003 paper [34], Asa Ben-Hur and Douglas Brutlag introduced their eMOTIF kernel for detecting protein homologues. This kernel gives a sequence similarity measure based on the motif content of a pair of sequences. A sequence x can in this context be represented in a vector space indexed by a set of motifs M as $\Phi(x) = (\phi_m(x))_{m \epsilon M}$, where $\phi_m(x)$ is the number of occurrences of the motif m in x. The motif kernel is then defined as a linear kernel over the motif contents: $K(x, x') = \Phi(x) \cdot \Phi(x')$ In most cases a motif appears only once in a sequence so this kernel essentially counts the number of motifs that are common to both sequences.

The motivation behind a kernel based on discrete sequence motifs is that because of their conserved nature, motifs can be used as a similarity measure even if the sequences are so distantly related that no sequence similarity outside the motifs are found [36]. Unlike other general string kernels and similarity search methods that weigh every position in a sequence equally, the motif kernel focus on parts of the sequences that are most conserved.

There are several databases available containing protein motifs that can be used for such a motif kernel. The eMOTIF kernel uses the eBLOCKS database [50]. This database uses PSI-BLAST [21] in a systematic fashion. For every protein sequence in the SwissProt database [51], a PSI-BLAST query is generated and run against SwissProt. The results are clustered and groups made that share the same levels of similarity. These groups are aligned and trimmed into blocks. The eMOTIF kernel extracts motifs from eBLOCKS using the eMOTIF method [36]. This method makes it possible to construct motifs with variable specificity and sensitivity. Table 3.2 provides some examples of eMOTIFs. In addition to ordinary amino acid characters, the motifs can also have alternative characters at each position, including a wildcard that will match any amino acid.

In this thesis, the eMOTIF content in the datasets is computed with a program provided by Asa Ben-Hur. We use the eMOTIFs in eBLOCKS version

Table 3.2: Examples of eMOTIFs.

$l.k.[kr]..l[eq]$
$[ast]..[iv][ilv]....[as]i[st]$
$[fwy].[filmv]..[filmv]...p.....e[filmv]$
$[ilmv]..[kr].l....[fwy]q[ilv]a[kqr]gm.[fy]las[kqr][kr].ihrdlaarnvlv.[de]..$ $[iv][ilmv]ki[as]dfgl[as]rd[iv]...[de]yy[kr]k..ngrlp[iv]kwma.e[as]l....y..[eq]$ $sdvws[fy]gvl[ilmv]wei[filmv]t.g$

1.0 of July 2002. This release contains over 520.000 different motifs. The program builds a trie data structure of these motifs, scans each sequence for the motifs by traversing this structure and outputs the motif hits for each sequence. We process the results and the kernel is computed by comparing the motif content of two and two sequences. While there are over 520.000 motifs in the eBLOCKS relase, the typical number of motif hits pr sequence is 100. Most motifs only occur once in a sequence. This means that the feature vectors of the sequences are very sparse.

3.2.2 Extending eMOTIF with additional positive motifs

The sparseness of the eMOTIF kernel suggests that homologous proteins will share very few motifs. As Ben-Hur reports, often a single motif is sufficient to classify a sequence [34]. But there are also many sequences that are not classified correctly, probably as a result of a lack of motifs shared between related sequences. This motivates the following experiment, where we try to increase the similarity information for the kernel by making additional motifs from the datasets. Each classifier for a SCOP superfamily or fold is given an additional set of motifs made to discriminately target the sequences of the superfamily/fold. The motifs are, using genetic programming, evolved as patterns that will match the proteins we want to recognise as coming from one specific superfamily/fold, while not match proteins in the others.

The program "GPboost" [52] is used for creating the motif patterns. In GPboost, each candidate solution is a variable length syntax tree written in the formal query language IQL [15]. The program uses sub-tree swapping crossover, tree generating mutation and reproduction as genetic operators and is trained with a positive and negative training set that in our case consist of labelled protein sequences. The IQL sentences produced are fed to the Interagon search hardware which quickly can find the number of matches in the data sets. The fitness of each IQL sentence is a function of the matches

in the positive and negative training sets.

The hardware supports several regular expression-like operators; we use only a small subset of these for building the motifs. Our solution language is formally defined in [52], but has previously been modified in the master thesis of Arne Johan Hestnes [53] to handle the protein alphabet of amino acids. At the basis, the solution language permits the use of amino acids and the wildcard operator that matches any amino acid. The pattern

$$GL.A$$

will for example match $GLAA$, $GLCA$, $GLDA$ etc. Patterns can be extended with the logical disjunction operator that returns a match if at least one of its two subpatterns finds a match. This makes it possible to give alternative amino acids at specific positions. The following pattern will match GLAA and GLCA:

$$GL\,(A|C)\,A$$

Finally, the Hamming distance operator specifies the minimum number of amino acids that must match in the pattern. The pattern

$$\{GLAA\} : (p \geq 3)$$

will for example match $GLAA$, $GLAC$ and $ELAA$, but for example not match $GLCC$ or $GCAQ$.

Note that the last operator makes it possible to specify that a pattern can have a certain number of mismatches. It is also possible to boost the importance of certain amino acid residues by using the Hamming operator in combination with the disjunction operator having the same residue as both inputs. For example, it is possible to double the weight of the Leucine residue in the current example so that a Leucine matching at the given position will count as two matches:

$$\{G\,(L|L)\,AA\} : (p \geq 3)$$

This pattern will for example match $GLAA$, $GLCC$ and $ELCA$.
The basic training sets made for GPboost include the same sequences as the training sets for the classifiers. But for each superfamily classifier, in addition to generate GPboost motifs with basis in the whole positive and negative trainingset, motifs will be made with one of the families left out of the positive training set. See Figure 3.2 for an example with three families. This is done to narrow the structural range each motif has to cover. Ten motifs are made for each such combination, so the total number of motifs made for each classifier will vary according to the number of families in the

superfamily.

For the fold benchmark, it is neither practical nor beneficial to generate motifs for all subsets excluding one superfamily because of the very high number of superfamilies pr fold. As we would like to base our kernel on approximately the same number of motifs as for the superfamily benchmark, we adopt a slightly different solution. The sequences of a fold are grouped into superfamilies and ten sets are made for the fold that each exclude one tenth of the sequences. All the motifs are created after running the GPboost on a population of 100 syntax trees for 50 generations. These parameters to GPboost are chosen based on earlier experience.

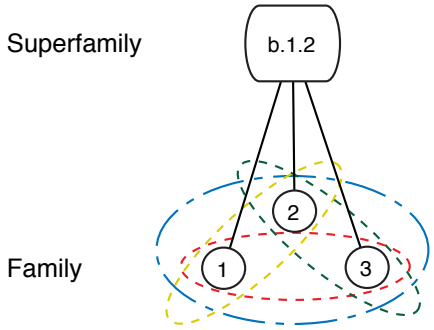

Figure 3.2: Subsets of the positive training set that each exclude one SCOP family are used when making GP motifs for the superfamily benchmark.

The IQL sentences generated with GPboost are matched against all of the protein sequences. This builds a motif matrix $A(M, N)$ of feature vectors which is the basis for computing the new kernel. The motif matrix contains a 1 at position (m, n) if motif m matches protein n, and a 0 if not. A preliminary linear kernel is then computed from this matrix by taking the dot product between two and two vectors. Since the sum of any two valid kernels is a valid kernel [10], the preliminary kernel is added to the earlier computed eMOTIF kernel and the resulting new kernel is used to train and test the support vector machine.

3.2.3 GP kernel

The excellent speed of the Interagon pattern matching chip allows us to take the previous idea a step further. Instead of just making positive motifs from the positive training set, the entire motif kernel can be based on motifs evolved using genetic programming. For each classifier, we generate motifs for all superfamilies, or for all folds when using the fold dataset. The

idea is that such a motif kernel, where each motif is trained to discriminate between protein domains of a given superfamily/fold and others, will give a much better discriminative representation of the protein sequences for a support vector machine, than if we for example only use motifs trained to discriminate the positive and negative sequences.

To get a correct validation of the performance, it is important to keep the test data untouched until testing. Since each classifier has different test sets, many different training sets for GPboost must be made that do not include data in the classifier's test set. This also includes the negative test set, so when making motifs for one superfamily, its positive training set does not include the classifer's negative test family in this superfamily and its negative training set does not include the classifier's positive and negative test set in other superfamilies. Also, the same method of leaving one family or one tenth of the sequences out, as used when making the GP extended eMOTIF kernel, is also used here on all training sets. The number of motifs generated for each classifier is constant. This means that the SVM classifiers for the superfamily benchmark will be based on 3350 different motifs. 3300 motifs are made for each classifier in the fold benchmark.

3.2.4 Genetic programming with boosting

Another opportunity available is to use the GPboost program to create boosted classifiers. While the GPkernel method use GP motifs as a basis for a kernel for a support vector machine, we would like to see how able boosted classifiers based on sequence motifs alone are to classify homologue sequences.

Each motif made previously gives a binary classification to every protein sequence. Using boosting, it is possible to combine several weighted motifs into one classification with extra attention given to the sequences that during training are hardest to classify correctly. We therefore build boosted motif classifiers that we train on the training set for 100 generations using a population of 100 in each generation and use 100 boosting iterations for each classifier. Each classifier will then produce discriminators for each protein sequence i that are the sum of the T weighted lesser hypothesises, $\sum_{t=1}^{T} \alpha_t \cdot h_t(i)$, where α_t is the weight of hypothesis h_t on protein sequence i, and T is the number of boosting iterations.

To further improve the accuracy of a classifier, majority voting is used. 10 sub-classifiers will be trained for each training set and the final discriminators for each sequence are the average of the 10 sub-classifiers. Figure 3.3 illustrates the setup of one GPboost classifier.

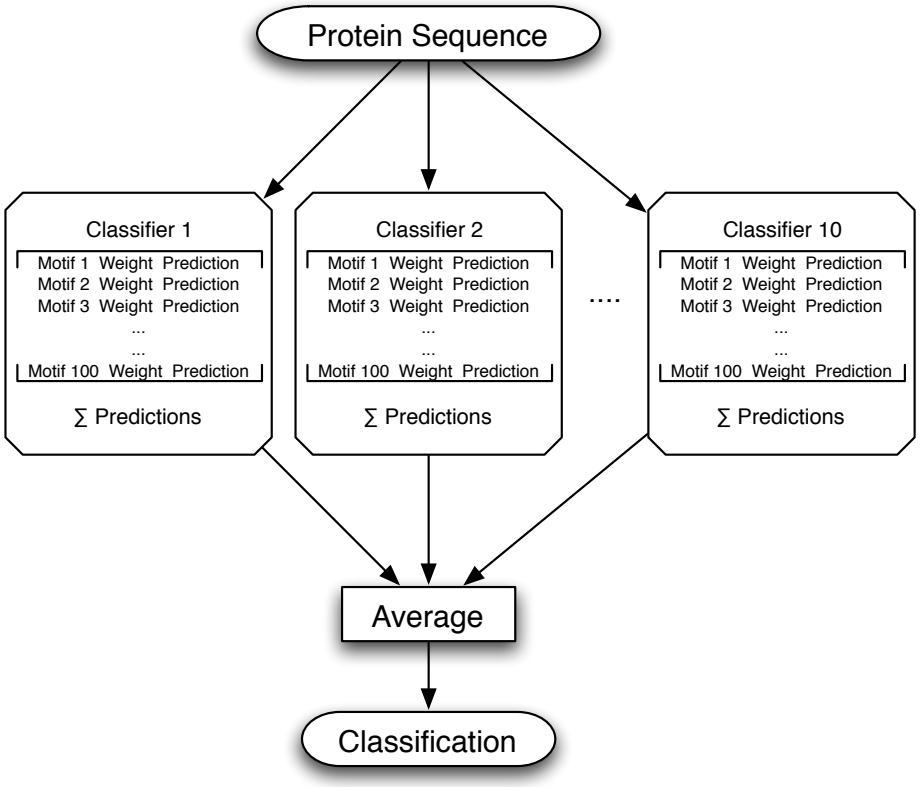

Figure 3.3: One boosted classifier consists of 10 sub classifiers that each are made from 100 boosted GP motifs.

3.2.5 The spectrum kernel

The spectrum kernel was presented in an article in 2001 [37]. This string kernel, while being conceptually simple and efficient to compute, was at the time shown to give reasonably good results despite its general nature. The basic idea is that two sequences can be said to be similar if they share many short subsequences, here called k-mers.

The k-spectrum of a of a sequence x, where $k \geq 1$, is the set of all k-length subsequences in x. The kernel is based on a feature representation where the dimension of a vector is equal to the number of all possible subsequences a of length k from the protein alphabet A. A feature map can then be defined on a sequence x of the input space X of all sequences to be

$$\Phi(x) = (\phi_a(x))_{a \epsilon A^k},$$

where $\phi_a(x)$ is the number of times that a occurs in x. The kernel is then defined as the linear kernel in this feature space of k-mers: $K_k(x, x') = \Phi_k(x) \cdot \Phi_k(x')$

A binary variant of the spectrum kernel is also possible. In this variant, $\phi_a(x)$ is equal to 1 if a occurs in x and otherwise equal to 0. This variant mentioned in the original article will not be used here. The original article also shows how the spectrum kernel can be computed efficiently using a suffix tree data structure. Because we in the current context are more interested in kernel performance than efficiency, the spectrum kernel is computed as a simple Python program that compares the k-mer content of two and two sequences and adds the dot product scores to a kernel matrix.

3.2.6 The mismatch kernel

The mismatch kernel [38] method has, since it was published in 2003/2004, been given a lot of attention as a benchmark method with which to compare other classifiers. The reason is its efficiency combined with a surprisingly good performance for being a general string kernel. The mismatch kernel is similar to the spectrum kernel described in section 3.2.5, except that each k-mer is allowed to have a given number of mismatches. This gives the possibility of using longer k-mer subsequences, thus increasing the information value of each feature.

The original variant of the mismatch kernel is computed efficiently using a mismatch tree structure, a variant of a suffix tree. We will instead use the Interagon pattern matching chip to compute the kernel. The Interagon Query Language makes it possible to search for patterns with an upper bound on the Hamming distance of the patterns. It is therefore possible to make a query of each k-mer and rapidly search the entire dataset for matches of this k-mer with a certain number of mismatches. The IQL query

$$\{abcde\} : p > 3$$

can for example be used to search for the pattern *abcde* with 1 mismatch.

Programming the pattern matching chip is facilitated by the PMC application programming interface. The API is fed IQL queries and uses call-back to a user-defined result processor when results are ready for processing. We do not compute the kernel explicitly using a matrix of feature values as this would yield a memory problem when dealing with a very high dimensional feature space (and also be inefficient). Instead, the results are stored in memory and when all the results for one k-mer are ready, the k-mer contri-

bution to the Gram matrix is calculated and added to the matrix.

If the number of queries in one batch is too high, the API will run out of memory. Therefore, we split the queries into batches of 10.000 and load these into the PMC when the results of the previous queries are ready. With this configuration, the single threaded C++ program for extracting k-mers from the dataset and computing a $(5, 1)$-mismatch kernel is finished in under 7 minutes when running on the first dataset of 4019 sequences, handling over 540.000 IQL queries.

3.2.7 PSI-BLAST and BLAST

The position-specific iterative BLAST [21] algorithm was released in 1997 and is one of the methods that may be called a standard method in remote homology detection. The algorithm is an extension of the more familiar BLAST algorithm. PSI-BLAST takes one sequence as input and, through iterative searches against a database, builds a profile of the input sequence using a multiple alignment of the highest scoring hits in each BLAST iteration. The profile is updated on each iteration, which means that PSI-BLAST can find additional homologs not found in previous iterations. In other words, the end result is a better sensitivity than standard BLAST.

There are several different variations on how to use PSI-BLAST experimentally for remote homology detection. Liao and Noble [28] used a random sequence in the positive training set and a multiple alignment of the same set as input. They then ran PSI-BLAST against the test set as a database for one iteration to align the test set to the training set profile. Another scheme is that by Kuang et al. [39]. Here, one profile is made for each sequence in the positive training set by searching against the NCBI non-redundant database [54]. The score reported for a test set is the average of all the scores for the profiles in the corresponding positive training set.

Because it is unclear to us which of these methods that actually gives the best results, we try both and also experiment with two different alternatives for combining classifiers. In the first method, we make profiles by letting Clustal version 1.83 create a multiple alignment of the positive training set and give this as input to PSI-BLAST. This gives one profile for each training set. We make a second set of profiles, one for each sequence in the data set, by running PSI-BLAST with each sequence as input for 10 iterations against the NCBI non-redundant (NR) database downloaded on 28. of March 2006.

The most important parameters to PSI-BLAST are the number of iterations and the two different E-value settings. The first E-value parameter sets the

threshold for the initial BLAST search while the second sets the threshold for the sequences that will be automatically included in the calculation of the profile. We use the standard settings (e=10.0 and h=0.002) for both methods.

To align sequences to a profile, a BLAST format database is generated from our data sets and PSI-BLAST is run with a profile and the database as input for 1 iteration. The E-values are extracted from the output and used as basis for discriminants to rank the test sequences when calculating the results. The diagram in Figure 3.4 shows how we conduct the experiments. We report results for four different PSI-BLAST and BLAST based methods:

I/ Using the Clustal based profiles and ranking a test set based on the alignment to the corresponding training set profile. This is the method most often used for PSI-BLAST.

II/ Using the NR profiles and reporting the average score resulting from aligning the positive training set profiles with the test set. This is the same method as used in [39].

III/ Using the NR profiles and reporting the score resulting from averaging the E-values made from aligning the positive training set profiles with the test set.

IV/ As method III, but with a pure pairwise BLAST search for each positive training set sequence against the test set. BLAST is run with default parameters expect for the expectation value threshold of 0.1

3.2.8 SVM-Pairwise

SVM-Pairwise [28] was the second method to use the support vector machine and was for a time the method yielding the best performance of all methods. SVM-Pairwise uses a direct comparison with each sequence in the classifier's whole training set to make the set of features for a support vector machine. The original method used the E-value of the Smith-Waterman algorithm [17] to compare a sequence to another. We make two versions of SVM-Pairwise. In the first, we gain a speedup for a minor loss in accuracy by using the E-value from BLAST pairwise alignments instead of using Smith-Waterman. Previous experiments have shown that using BLAST gives approximately the same accuracy as using Smith-Waterman for SVM-Pairwise [34]. BLAST is run with default parameters expect for the expectation value threshold of 0.1.

For the second version of SVM-Pairwise we utilise the second set of profiles made for the PSI-BLAST experiment. Instead of aligning a sequence against

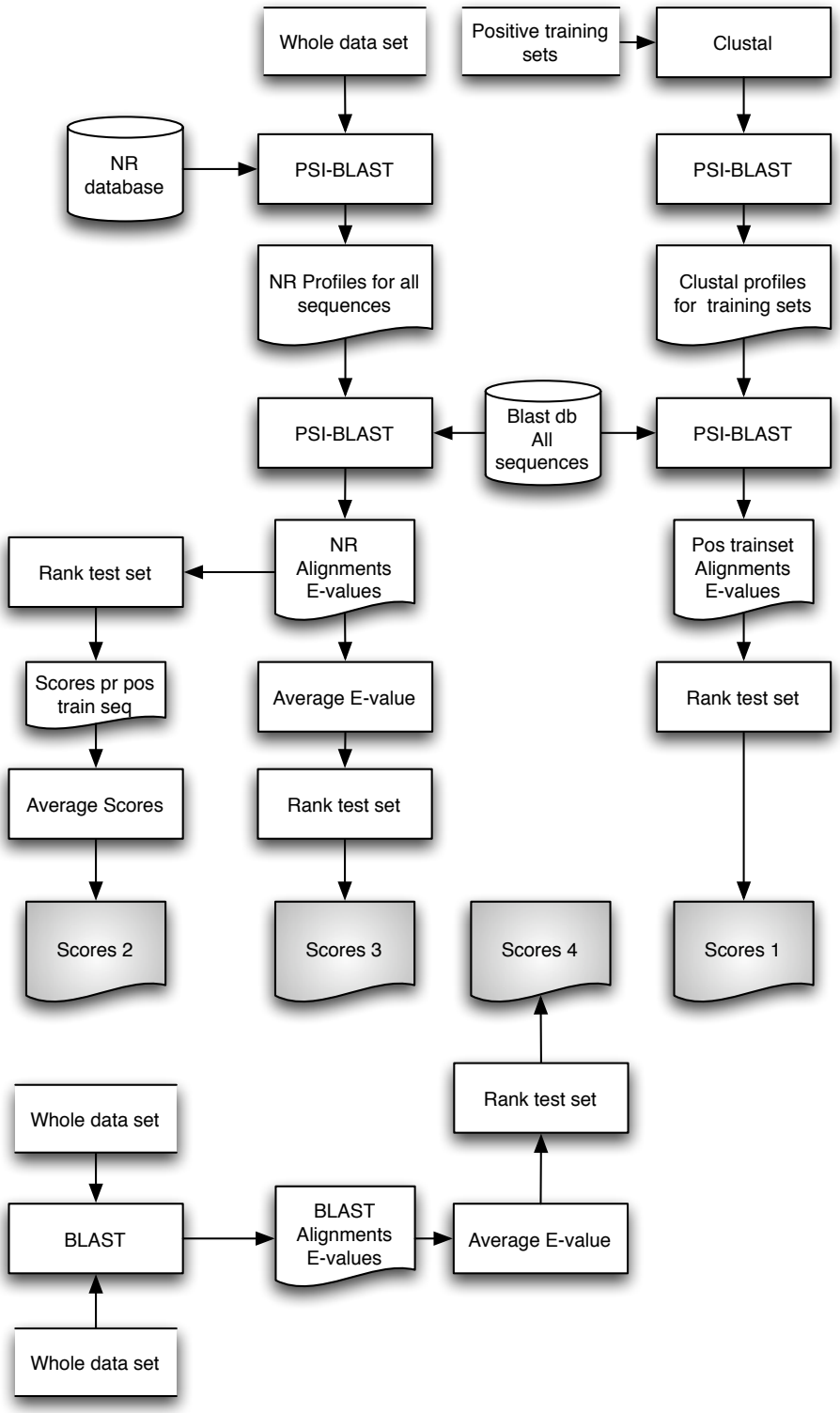

Figure 3.4: The rather extensive PSI-BLAST and BLAST experiments.

the sequences of a training set, a sequence is aligned to the NR profiles of the training set. The motivation for doing this is to see what can be gained by using profiles; it is expected that more and better alignments can be made to the profiles, giving a more accurate classifier. For both versions, the negative logarithm of the E-values are used as features.

The feature vectors obtained are used with the radial basis kernel. This kernel is defined as

$$K(x, y) = e^{-\frac{\langle x,x\rangle - 2\langle x,y\rangle + \langle y,y\rangle}{2\sigma^2}} + 1$$

Here, $\langle x, y\rangle$ means the linear kernel (the dot product of the feature vectors) and σ is the median of the distance from each positive training point to the nearest negative training point. The last is a heuristic for setting the width of the radial basis kernel. An asymmetric soft margin is implemented by adding to the diagonal of the kernel matrix 0.02 times the fraction of the training set with the same label as the current sequence. This is completely the same setup as the original by Liao and Noble [28].

3.3 Training and testing the SVM classifier

The Gist package [55] version 2.2 is chosen as the preferred support vector machine classifier. The Gist package contains software tools for support vector machine classification and for kernel principal component analysis. Gist makes it possible to compute a kernel matrix manually and give it as input to the classifier. The support vector machine requires several different kernel matrices pr family when training and testing. Instead of computing all these separately for each family, one big kernel matrix for all proteins is computed and the smaller matrices are made from this by looking up the right numbers from the big matrix using the sequences in the training and testsets.

Because there are many more negative examples than positives, the classifier's accuracy does not give a good measure of the performance. Instead, we use the area under the ROC curve [56], which is a plot of sensitivity vs 1-specificity for varying classification thresholds. It shows the trade-off between sensitivity and specificity and the area under the curve is a simplified summary of how sensitive and specific the classifier is overall. We also calculate the ROC50 score, which is equal to the area under the standard ROC curve, but where we only count true positives until the 50 first false positives are found. This gives a better measurement of the practical usefulness of a method than the standard ROC. To do a family by family comparison between the different classifiers, a non-parametric alternative to

the paired student's t-test, the Wilcoxon signed rank test [57] is used. The Wilcoxon test does not only count the number of better classifications as a sign test does, but also takes into account the value of the absolute differences between measurements and is not based on any assumption about the underlying distribution of differences. A p-value lower than 0.05 is considered a significant difference between compared methods.

Chapter 4

Results

This chapter presents the results of all experiments. The chapter is divided into four sections. In the first section, we look at how the motif-based methods compare to one another. We also look at some of the properties of the GPkernel. In the second section, we compare the best performing motif method, the GPkernel, with the string and alignment-based kernels. In the third section, we look at a specific data set and try to find out how the GPkernel achieves its good performance. In the final section, we look at some additional results of the competing methods.

The results are based on the ROC and ROC-50 scores that the classifiers achieve on the remote homology detection benchmark and the fold detection benchmark. We present the results as the cumulative number of test sets for which we achieve a score higher than a given value. Some additional results are also reported in appendix A.

4.1 The GPkernel has the best performance of the motif methods

The motif-based methods, GPkernel, eMOTIF, GPextended and GPboost, have huge differences in performance on the two benchmarks. Figures 4.1 and 4.2 show the performance of the motif-based methods on the superfamily benchmark. The figures show that the GPkernel has the best performance of the motif methods. On the ROC scores, the GPkernel is significantly better than eMOTIF, with a p-value of $6.95 \cdot 10^{-5}$ on a Wilcoxon signed rank test. On the ROC-50 score, the GPkernel is slightly better than the eMOTIF kernel, with a p-value of $1.27 \cdot 10^{-1}$ on a Wilcoxon signed rank test.

Judging by the ROC-50 scores, the eMOTIF kernel does not seem to benefit from extra positive GP motifs on the superfamily benchmark. The GPex-

tended kernel actually perform a bit worse than the eMOTIF kernel on some of the best classified families and the eMOTIF is perhaps slightly better (p-value $3.19 \cdot 10^{-1}$). But if we look at the whole ROC curve, we see that the GPextended is better than eMOTIF (p-value $5.37 \cdot 10^{-2}$). The performance gain by adding extra GP motifs to the eMOTIF kernel is smaller than expected. The small differences between GPextended and eMOTIF gives a hint about the discriminative properties of the GP motifs and suggests that the eMOTIF kernel has a sufficient set of motifs for the superfamily benchmark. It would require many more GP motifs to further improve the results of the GPextended kernel. The boosted classifiers have a poorer performance than the kernel-based classifiers. No true positives are found before 50 negatives for 20 of the families on the ROC-50 score, but the boosted classifiers show the same tendency as the other motif-based methods.

Figures 4.3 and 4.4 plot in a similar way the performance on the fold benchmark. All methods show some drop in performance, compared with the superfamily benchmark. Most evident is the performance drop for the eMOTIF kernel. Because most of the eMOTIFs are relatively specific, the sequences that belong to a fold will on average share few eMOTIFs, giving a very sparse kernel. As will be discussed in section 4.3, this might explain the huge performance drop for the eMOTIF method compared with its performance on the superfamily benchmark.

The GPextended method adds several GP motifs to each classifier and this gives a much better performance on the fold benchmark. There is substantial improvement in ROC and ROC-50 performance (p-values $3.30 \cdot 10^{-5}$ and $7.23 \cdot 10^{-5}$ respectively) on the fold benchmark with the additional set of GP motifs compared to the eMOTIF kernel. According to the ROC-50 scores, the GPboost method is not able to classify 20 of the superfamilies to the correct fold. Except for one, these 20 superfamilies belong to different folds than the 20 families not correctly classified by GPboost in the superfamily benchmark. Though it might seem otherwise from the figures, according to a Wilcoxon signed rank test there is little difference between GPboost and eMOTIF on ROC score (p-value 0.15) and the eMOTIF is better on ROC-50 score ($5.26 \cdot 10^{-2}$). On a sign test however, GPboost is better than eMOTIF on ROC score (0.08) and significantly better on ROC-50 score ($5.16 \cdot 10^{-3}$).

The GPkernel has a very good performance on fold detection. With a unique set of motifs made for each fold, it has a better basis for discriminating between sequences of different folds. Figure 4.5 gives an idea of how important the motifs trained on the negative sequences are for the GPkernel classifier. If we test the GPkernel with an equal number of positive motifs only, the average ROC-50 score on the superfamily benchmark falls by 30%, but the results vary from family to family. The importance of the negative motifs

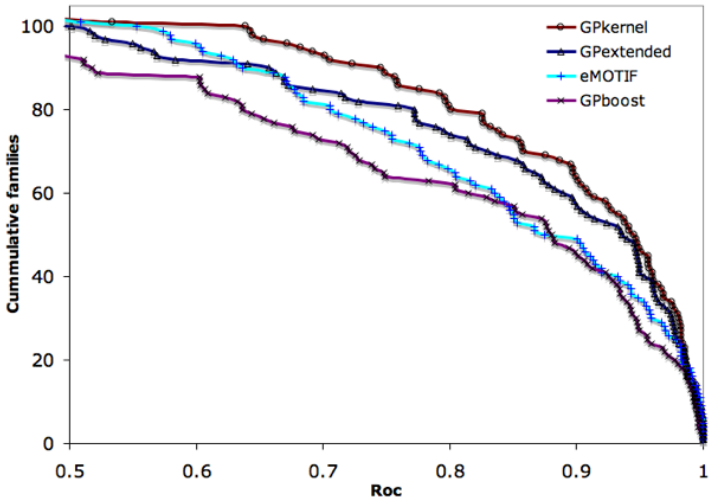

Figure 4.1: ROC results for motif methods on superfamily benchmark.

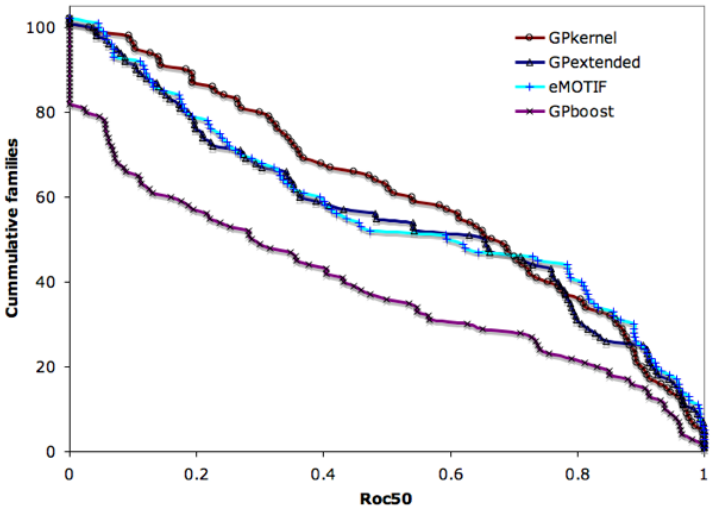

Figure 4.2: ROC-50 results for motif methods on superfamily benchmark.

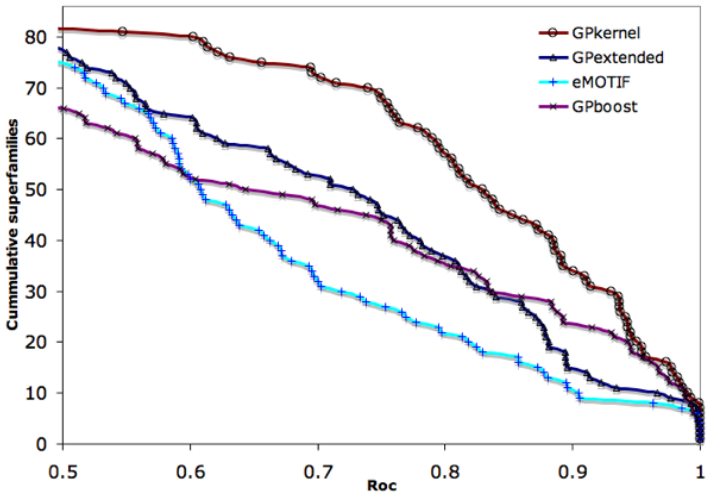

Figure 4.3: ROC results for motif methods on fold benchmark.

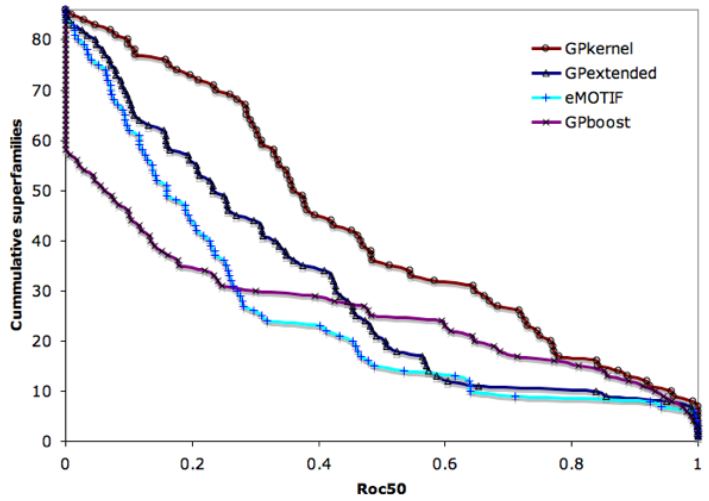

Figure 4.4: ROC-50 results for motif methods on fold benchmark.

is also evident in the performance differences between the GPkernel, the GPboost and GPextended methods. The boosted classifiers base their predictions only on motifs trained to match the classifier's positive training set, and the GPextended method also adds only positive GP motifs. Both methods are much less accurate compared to the GPkernel.

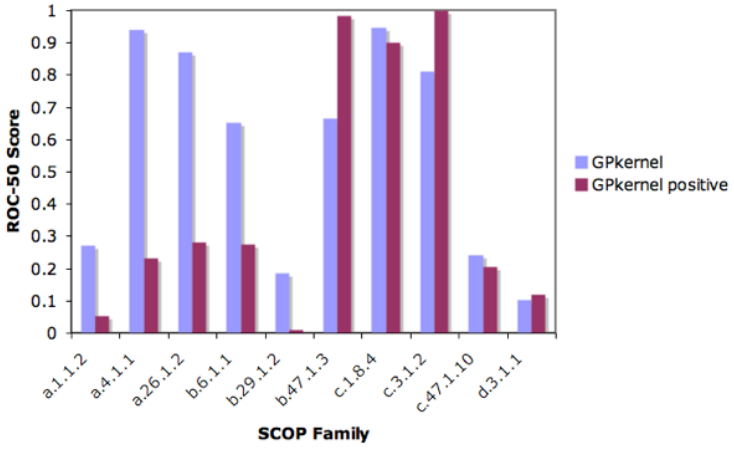

Figure 4.5: A GPkernel trained with only positive motifs will on average lose discriminative power, but the result varies from family to family. The figure shows some example families taken from the superfamily benchmark.

4.2 The GPkernel is best on the most difficult fold benchmark

Figures 4.7, 4.6, 4.8 and 4.9 show the performance of the GPkernel, in comparison with the eMOTIF, Mismatch, Spectrum, and SVM-Pairwise kernels and PSI-BLAST. The SVM-Pairwise kernel is based on E-values from pairwise BLAST alignments and the PSI-BLAST shown is based on average E-values obtained from using the profiles made by searching the NCBI nonredundant database. The Mismatch kernel has been computed with k-mers of length 5 and with 1 mismatch, the Spectrum kernel with k-mers of length 3. Results for Spectrum kernels of other k-mer lengths can be found in appendix A. Results from the other variants of SVM-Pairwise, PSI-BLAST and BLAST are discussed in section 4.4.

On the superfamily benchmark, GPkernel has the best overall performance. It has significantly better ROC scores than the other methods (all p-values less than $8.4 \cdot 10^{-5}$ on Wilcoxon signed rank tests). GPkernel also has better ROC-50 scores than eMOTIF (p-value $1.27 \cdot 10^{-1}$), SVM-Pairwise

($1.23 \cdot 10^{-1}$), and PSI-BLAST ($9.34 \cdot 10^{-2}$) and has significantly better ROC-50 performance than the Spectrum kernel ($2.65 \cdot 10^{-10}$) and the Mismatch kernel ($3.36 \cdot 10^{-3}$). On a sign test, the GPkernel is also significantly better than SVM-Pairwise (p-value 0.05) on ROC-50 performance).

Figures 4.8 and 4.9 show how the methods compare on the fold benchmark. As would be expected, there seems to be a bigger difference between the methods when the level of sequence similarity is very low. The BLAST-based methods have a hard time producing effective alignments between sequences related at the fold level. The performance of SVM-Pairwise is poor compared with its performance on the superfamily benchmark. For PSI-BLAST, only a fourth of the test sets have a ROC-50 score higher than 0.2. Clearly PSI-BLAST is not suited as a method for fold detection. The Mismatch kernel has a stable performance on both benchmarks. The GP-kernel has the best performance on the fold benchmark, with significantly better ROC-50 scores than the second best performing Mismatch kernel (with p-value $5.57 \cdot 10^{-4}$ on a Wilcoxon signed rank test). The GPkernel is also significantly better than SVM-Pairwise ($2.81 \cdot 10^{-8}$) and PSI-BLAST ($1.80 \cdot 10^{-11}$) on ROC-50 scores. The GPkernel has significantly better ROC scores than all other methods, with highest p-value in comparison with the Mismatch kernel ($2.72 \cdot 10^{-7}$).

4.3 General motifs are beneficial for fold detection

One of the SCOP superfamilies (b.68.1) that participate as a test set in the fold detection benchmark is classified well by the GPkernel method (ROC-50 score of 0.903) but achieves a lower score with the eMOTIF method (0.128). Even though there seems to be a mild correlation (0.16) between the number of eMOTIF matches for a fold and the ROC-50 score achieved, the training and test sets for this superfamily do not have significantly fewer eMOTIF matches than other training and test sets. More important is the number of eMOTIFs shared between sequences. This number varies a lot between different pairs of sequences. By simple inspection of the eMOTIF kernel matrix, it can be seen that most sequences share zero or very few eMOTIFs.

If we calculate the average number of eMOTIFs shared between sequences, we find that the sequences in the b.68 fold on average share 0.73 eMOTIFs. This is barely more than the average number of eMOTIFs shared between two random sequences in the dataset (0.52). It is less than the average number for sequences within a fold (2.41) which again is much less than the average shared between sequences of a superfamily (11.92). This shows that because sequences at the fold level have a very low sequence similarity, and

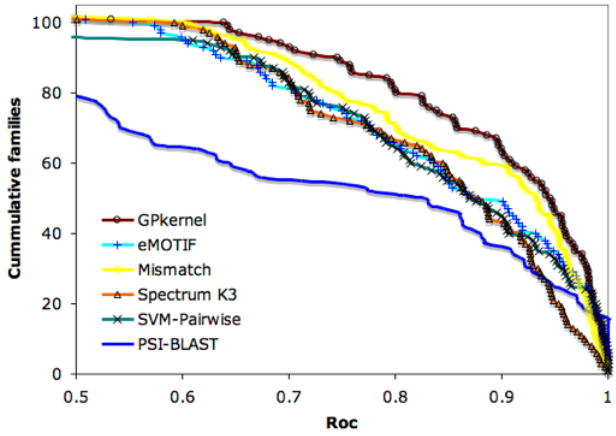

Figure 4.6: ROC results on superfamily benchmark.

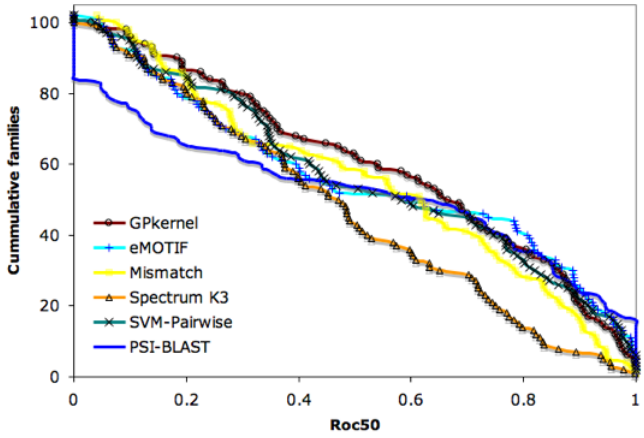

Figure 4.7: ROC-50 results on superfamily benchmark.

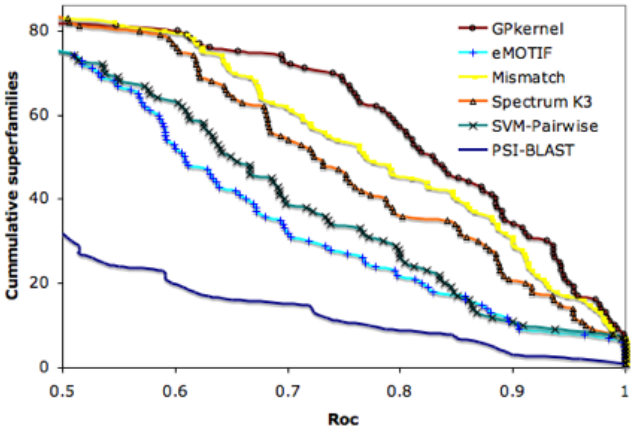

Figure 4.8: ROC results on fold benchmark.

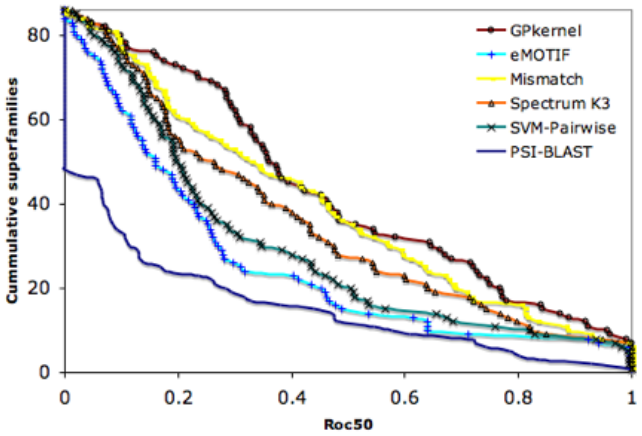

Figure 4.9: ROC-50 results on fold benchmark.

because of the specificity of most of the eMOTIFs, the number of eMOTIFs shared between sequences in a fold will also be low. There is a correlation (0.20) between the average number of shared eMOTIFs in the training sets and the ROC-50 score achieved on the superfamily benchmark, but to our surprise, on the fold benchmark the same correlation is negative (-0.12). Because of the very small number of averaged shared eMOTIFs for the fold data, the fold correlation might be more inaccurate than the superfamily correlation. Assumably, the low number of shared motifs in the b.68 fold is the biggest factor that influences the classification result.

Table 4.1 shows examples of GP motifs trained on the training set for the b.68 fold classifier. The GP motifs do in general not share any huge similarities with the eMOTIFs that match the sequences of the fold. For each motif, the table shows the percentage of sequences matched in the training and test sets. The positive training set has 12 sequences, the negative 3590 sequences. The positive test set also has 12 sequences, the negative 226 sequences. The GP motifs do match a higher percentage of the positive sequences than the negatives, but the considerable number of negative sequences that are matched shows the difficulty of finding simple discrete sequence motifs that cover many sequences of a fold while also being as specific as possible. The best GP motifs tend to be either very short sequences or very long complicated expressions with multiple alternative amino acids at each position in the motif.

Table 4.1: Examples of GP motifs for b.68 fold classifier.

Motif	**PTr**	**NTr**	**PTe**	**NTe**
$\{MEEIEII : p >= 3\}$	67	41	67	55
$\{IQIIIEE : p >= 3\}$	83	38	92	50
$\{(I\|I)E(E\|(I\|E)) : p >= 4\}$	58	37	83	51
$\{TQ(D\|H)(K\|C)(D\|H)((((D\|H)\|A)\|A)\|A)$ $TQ((H\|A)\|A)TQ((D\|H)\|A)(I\|A) : p >= 7\}$	33	20	33	23
$\{M(L\|L)CARACAARAA(L\|L)RACAA : p >= 6\}$	8	28	50	44
$\{AALAALA(A\|M)AA.ILAL(A\|M)AA(C\|M)$ $AV.IL(.\|T)A.ILAAALA(.\|(A\|M))V.IL$ $VAA.ILL(.\|T).IA(A\|M)AALA$ $(A\|M)V.ILV(R\|M) : p >= 20\}$	50	28	25	45
$\{(L\|(M\|A))(L\|(L\|(M\|A)))(L\|(M\|A))(L\|((L\|A)$ $\|A))M : p >= 5\}$	83	37	67	54

When looking at all GP motifs made, we find that each motif on average matches nearly a fourth of all sequences. All sequences will therefore share many GP motifs, which means that there will be many "erroneous" motif predictions. To achieve good accuracy with ensemble classifiers, that is, classifiers that combine several predictions (motifs), it is important that the whole solution space is covered and that the erroneous predictions do not correlate too much so that the level of noise is kept to a minimum. To test the level of correlation, we compute all possible correlations between positive motifs on negative sequences. We then check for correlation between the average correlation for each classifier and the ROC-50 score it has obtained. Oddly, we then get a weak positive correlation (0.13) which means that the amount of correlation for motif matches on negative sequences does not determine the result of the GPkernel.

If we compare the related sequences of a superfamily or fold with randomly chosen sequences, we find that related sequences share more motifs than randomly chosen sequences. On average, sequences in a superfamily have a higher correlation in their motif matches than other sequences. This includes both the positive motifs made to match the given classifier's positive training set and the other motifs. On the positive motifs, the average correlation coefficient for related sequences is 0.25 versus only 0.16 for unrelated sequences and on the other motifs the average correlation coefficient is 0.33 for related sequences versus 0.21 for unrelated sequences. Related sequences therefore share more of all available motifs than unrelated sequences, explaining the GPkernel's performance.

Another kernel that also has a good performance on fold detection is the mismatch kernel. This kernel is based on a much larger feature set of even more unspecific patterns than the GPkernel. For the mismatch kernel, the generality of the patterns ensures that the whole solution space of sequences is covered and that most sequences share at least a few patterns. The GPkernel achieves good coverage by training a certain amount of motifs pr superfamily or fold. The GP motifs, while not being too specific, are still more tuned to discriminate between sequences of different folds than the completely general mismatch k-mer patterns. This suggests that to capture the small sequence similarity that exists at the fold level, motif-based classifiers may benefit from motifs that are general enough to match a significant number of the weakly similar sequences of a fold.

4.4 Averaged-BLAST is almost comparable to SVM-Pairwise

PSI-BLAST has been used as a reference method in many other experiments in the past [28, 32, 38, 39, 43]. It is not straightforward to compare PSI-BLAST, which is an iterative method that requires a single sequence as input, with other methods that require multiple input sequences. One of the most common approaches is to use a multiple alignment as input, but as seen in Figures 4.10 and 4.11, this gives poor results on our two benchmarks. We have therefore investigated other possibilities for finding homologues with PSI-BLAST and other alignment-based methods.

Figures 4.10 and 4.11 show how all the different alignment-based methods explained in sections 3.2.7 and 3.2.8 compare to one another. It can be seen that most of the PSI-BLAST alternatives show poor performance, including both the Clustal-based profiles and the profiles made from the NCBI non-redundant (NR) database. It has been reported [39] that SVM-Pairwise based on alignment to multiple PSI-BLAST profiles might give better results than using pairwise Smith-Waterman or BLAST alignments. Our results do not support this hypothesis. We do get 5% more pairwise alignments with NR profiles than with pure BLAST, but still the BLAST-based SVM-Pairwise is better than the profile-based SVM-Pairwise on the superfamily benchmark (p-value $3.85 \cdot 10^{-1}$) and significantly better on the fold benchmark (p-value $1.90 \cdot 10^{-8}$) .

The NR profiles are made by running PSI-BLAST for ten iterations against the NCBI non-redundant database. While profile models should give a better representation of the similarities and variance in related sequences, it is possible that the resulting profiles will tend to be too equal and general to produce good alignments to the whole solution space of homologues. By looking at some of the alignments made between NR profiles and sequences, we find that sometimes, other related sequences achieve a better alignment to a NR profile than the sequence that was used as the starting point for creating the profile. The profiles might gravitate to the centre of the majority and align worse to the related sequences that differ from the majority.

The two benchmarks used in this thesis are much larger than previous benchmarks. We note that most of the recent publications have stopped using PSI-BLAST as a benchmark method and we do not know of any extensive tests on PSI-BLAST as a method for fold detection. It is possible that our larger data sets give a more proper estimate of the method's capability.

Figures 4.10 and 4.11 show the results of two new methods, one using the

average BLAST E-value of alignment against the positive training set and the other using the average PSI-BLAST E-value to classify sequences. While SVM-Pairwise is significantly better than Averaged-BLAST on both the superfamily and fold benchmarks (p-values $1.39 \cdot 10^{-3}$ and $6.21 \cdot 10^{-3}$ respectively), the Averaged-BLAST method does not require training an SVM. The reason for the good results is that, given standard parameter settings, BLAST will usually not be able to align unrelated sequences. Even if BLAST is able to align only one of the sequences of the positive training set to an unknown sequence, the sequence will be reported as positive, but with a low value discriminant. This gives an opportunity to get many false positives, but by averaging all BLAST E-values, the true positives will dominate with higher discriminants. With a suitable classification threshold, the method can give good results. The PSI-BLAST version with average E-values is the PSI-BLAST method getting the best results on our benchmarks, but is still poorer compared to average BLAST.

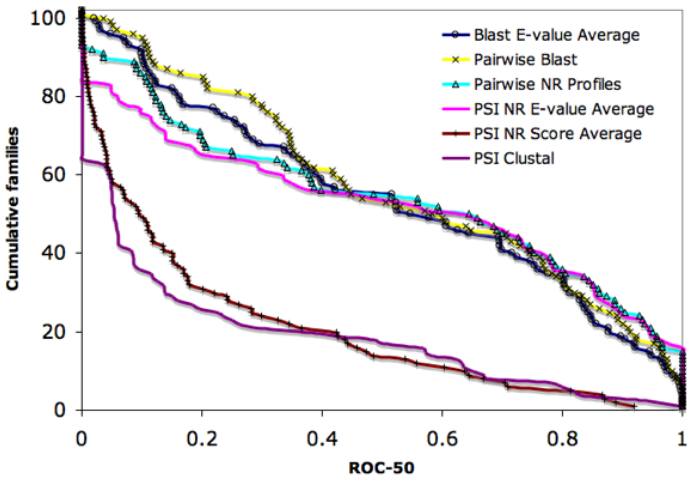

Figure 4.10: ROC-50 results on superfamily benchmark for alignment-based methods.

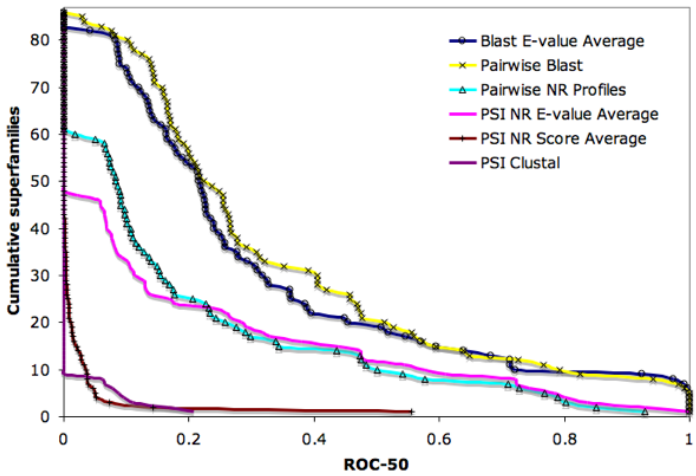

Figure 4.11: ROC-50 results on fold benchmark for alignment-based methods.

Chapter 5

Discussion

This chapter gives an additional discussion of the results obtained in the experiments.

5.1 The fold benchmark is considerable harder than the superfamily benchmark

We have used two benchmarks based on the SCOP database [48] in this thesis. The superfamily benchmark measures a method's ability to recognise remote homologues, the fold benchmark measures a method's ability to assign sequences into correct folds. Proteins are defined as having a common fold in SCOP if they have the same major secondary structures in the same arrangement and with the same topological connections. Unlike the majority of the sequences of a SCOP superfamily, proteins placed together in the same fold category may not have a common evolutionary origin and their sequence similarities are very small. This makes fold detection a considerably tougher benchmark than the superfamily benchmark.

Both our benchmarks are extensive and test almost twice the number of families and superfamilies than earlier used benchmarks [47]. We also require that there be at least ten sequences in both the positive training set and the positive test set, so that each classifier is thoroughly trained and tested. This should give a proper view of the performance of each method. Table 5.1 shows the average ROC and ROC-50 scores for all the methods. It can be seen from the table that the GPkernel has the highest and most stable performance on both benchmarks.

Figure 5.1 illustrates the differences in ROC-50 scores between the methods on the two benchmarks. All methods have a better average ROC-50 score on the superfamily benchmark than on the fold benchmark, but it is interesting to see how some methods show much larger performance differences

Table 5.1: Average ROC and ROC-50 scores of all methods.

Method	Superfamily level		Fold level	
	ROC	ROC-50	ROC	ROC-50
GPkernel	0.902	0.590	0.844	0.514
GPextended	0.869	0.542	0.753	0.371
GPboost	0.797	0.375	0.688	0.298
eMOTIF	0.857	0.551	0.698	0.308
Mismatch	0.878	0.543	0.814	0.467
Spectrum K3	0.858	0.477	0.788	0.421
SVM-Pairwise Blast	0.849	0.555	0.724	0.359
SVM-Pairwise NR	0.860	0.527	0.608	0.182
PSI Clustal	0.575	0.175	0.501	0.010
PSI NR score avg	0.558	0.128	0.506	0.016
PSI NR E-val avg	0.747	0.505	0.569	0.172
BLAST E-val avg	0.830	0.527	0.708	0.332

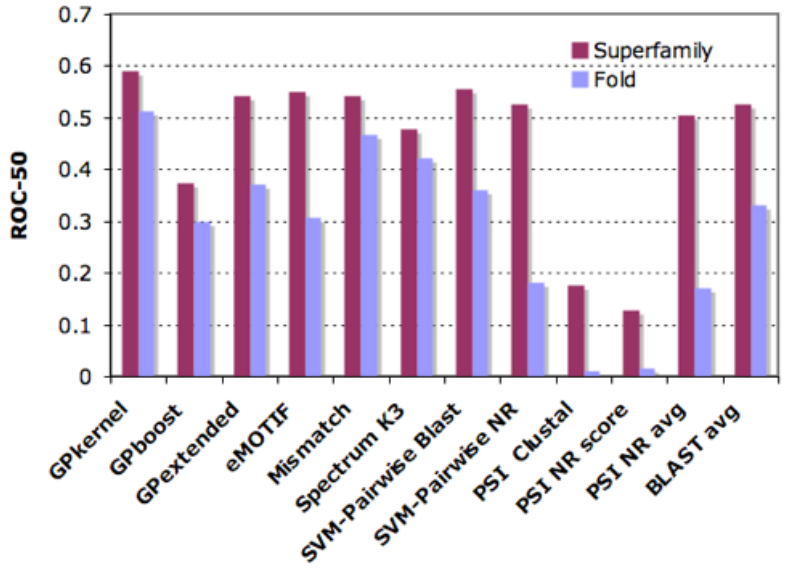

Figure 5.1: Average ROC-50 results for all methods on superfamily and fold benchmarks.

between the two benchmarks than others. The general string based kernels, Spectrum and Mismatch, have small differences. This is also true for the GP motif based methods, which have relative general motifs. The string kernels and the GPkernel are also the methods with the best performance on the fold detection benchmark. This suggests that capturing the tiny sequence similarities at the fold level requires a broad set of general patterns.

It is also easy to see how the alignment based methods fail on the fold benchmark. BLAST and PSI-BLAST do not seem to be able to align sequences at this level, even with E-value thresholds of 10.0. PSI-BLAST gives especially poor results. In the Clustal version, the positive training set is aligned with Clustal before given as input to PSI-BLAST. This should result in a good profile description of the training set sequences. Indeed, on the superfamily benchmark, there is a small positive correlation (0.10) between the size of the positive training set and the ROC-50 score achieved. It might be that the positive training sets from the fold benchmark, having a large group of relatively dissimilar sequences, will give a "diluted" profile. We note that on the fold benchmark, the Clustal based PSI-BLAST was very often not able to align a single sequence in the test set to a given profile that was based on the alignment of the positive training set of a fold. One might argue that using Clustal first may not give the best impression of PSI-BLAST performance, but from the other PSI-BLAST results we can still conclude that the method does not seem suited for fold detection.

Earlier work [58] has reported an average ROC score of 0.675 on Clustal based PSI-BLAST for a superfamily benchmark based on 54 families from SCOP version 1.53. This is an average ROC score that is 17% higher than our average ROC score on 102 families from SCOP 1.67. Liao and Noble reported [28], using the same method on the same SCOP 1.53 benchmark, that of 54 families, 30 have a ROC score higher than 0.5 and 10 have a ROC score higher than 0.7. Of our 102 families, 40 families have a ROC score higher than 0.5 and 19 families have a ROC score higher than 0.7. We therefore suspect that our benchmark based on SCOP 1.67 must be harder than the previous 1.53 benchmark.

5.2 Motif based methods perform well on homology detection

To recognise remote protein homologues from amino acid sequences means to compare sequences for similarities that often are so weak that their similarities can mistakenly be taken for random similarities. It is therefore a sound idea to investigate similarity measures that focus on the most con-

served regions in sequences, the motif regions, since these are the regions where homologue sequences are most likely to be similar. It has earlier been claimed that the eMOTIF kernel gives very good results on remote homology detection [34], something our results seem to support. The eMOTIF approach to making motifs makes it possible to generate motifs with variable sensitivity and specificity [36], but most of the motifs are still relatively specific. The eMOTIF kernel we have computed is based on over 520.000 eMOTIFs, but despite the abundance of eMOTIFs, several sequences are not correctly classified by the kernel. This is especially true for the fold benchmark. This has motivated our effort to produce a better set of motifs for a motif based kernel.

This thesis has presented methods for remote homology detection and fold detection founded on automatic motif synthesis. Our main contribution, the GPkernel, is a similarity measure based on the motif content of two sequences where the motifs are generated by hardware accelerated genetic programming. This represents a very different approach to motif synthesis than the eMOTIF method. The GP motifs are trained on both positive and negative training sets to distinguish related sequences from unrelated sequences. The genetic programming process might find motifs that other motif synthesising methods do not find. A search in Interpro [59] with some of the sequences of the b.68 fold did not give any obvious similar motifs to the GP motifs shown in Table 4.1. As discussed in section 4.3, the GP motifs tend to be quite unspecific compared to for example Prosite patterns [23] and perhaps it therefore would be more precise to refer to the GP motifs as patterns and not as motifs.

The Mismatch kernel is computed efficiently by the Interagon pattern matching chip and the method has good performance on both the superfamily and the fold benchmark. While some of the GP motifs might appear similar to mismatch k-mers, the GP motifs are more than just mismatch k-mers with syntactic sugar. The GPkernel has better ROC-50 performance than the Mismatch kernel on 63 of 102 test families on the superfamily benchmark and is better on 55 of the 86 test superfamilies of the fold benchmark. The GPkernel is based only on the motifs that the GP algorithm finds to give the best description of the different superfamilies and folds. Combining the GP motifs with a discriminative classifier like the support vector machine, gives a powerful discrimination-based learner. Like already seen in the previous chapter, the GPkernel is able to correctly classify a higher number of new sequences to the correct superfamily and to the correct fold than the other methods.

5.3 The GPkernel can still be improved

While noise reduction and feature extraction, such as Latent semantic analysis (see appendix A), might be a possibility for the GPkernel, there is more work that could be done directly on the GPkernel as a method. The GP motifs are currently made from training sets that span almost an entire superfamily or fold. We have tried to reduce the taxonomical distance by leaving one family out for the superfamily benchmark and leaving one tenth of the sequences out for the fold benchmark motifs. It could be interesting to see if much smaller positive training sets would produce more specific GP motifs, and what this could mean for the performance of the method.

Another thing that could improve the GPkernel is to use boosted motifs. The GPboost method gives good results compared with the other non-SVM based methods. It is expected that a GPkernel based on boosted motifs could give better classification accuracy. It could then be interesting to use our updated benchmarks to compare the results of a GPkernel based on boosted motifs with some of the most recent methods to enter the field of remote homology detection.

Chapter 6

Conclusion

This thesis has explored methods in remote homology and fold detection based on automatic motif synthesis. The GPkernel is a motif-based method that uses motifs evolved by genetic programming for remote homology and fold detection. Tested on two updated benchmarks, the GPkernel has significantly better ROC scores than all other competing methods and also has significantly better ROC-50 scores than the majority of the competing methods.

The motifs trained on the different classes of negative sequences are important to the GP-kernel's predictive power. While the positive motifs match more positive sequences than negative sequences, they do not necessarily alone predict the superfamily or fold of a protein. But combined with the relative absence of negative motifs from the positive sequences, this gives a better differentiation between related and unrelated sequences than many other methods. This is also visible when comparing the GPkernel with the related GPboost method that has a good set of motifs, but lack the extra discriminative power of a support vector machine trained with both positive and negative motifs that capture different aspects of the training set. While each single motif alone will not discriminate between different sets of homologues, overall, related sequences will share more motifs than unrelated sequences.

Most of the motifs that result from the genetic programming process show relatively little specificity, but not having too specific motifs seems to be beneficial on the fold detection benchmark. The nature of the genetic programming process also means that some of the motifs produced will act more as noise than contribute to sequence discrimination. This means that the GPkernel could be improved by techniques for feature extraction and noise removal. A preliminary study of noise removal on the GPkernel is given in appendix A.

It is still possible to improve the GPkernel. For example could boosting be used to improve the accuracy and discriminative power of the motifs. Another opportunity could be to combine different types of kernels. No single method works best in all cases. It could therefore be an idea to combine for example alignment based kernels with motif based kernels. In that case, the GPkernel would be a natural representative for the motif based kernels.

Bibliography

[1] Berg Jeremy, Tymoczko John, and Stryer Lubert. *Biochemistry Fifth edition.* W.H. Freeman and Company, 2002.

[2] Christian B. Anfinsen. Principles that govern the folding of protein chains. *Science*, 181:223–230, 1973.

[3] National Humane Genome Research Institute. Protein stucture illustration. www.genome.gov//Pages/Hyperion/DIR/VIP/Glossary/Illustration/protein.cfm.

[4] T. Mitchell. *Machine learning.* McGraw-Hill, New York, 1996.

[5] Prechelt Lutz. Early stopping-but when? *Neural Networks: Tricks of the Trade*, pages 55–69, 1996.

[6] R. Schapire. The boosting approach to machine learning: An overview. Nonlinear Estimation and Classification, 2003.

[7] Yoav Freund and Schapire Robert E. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pages 148–156, 1996.

[8] J.R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

[9] J. Holland. *Adaption in natural and artificial systems.* University of Michigan Press, 1975.

[10] Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, 2001.

[11] Cristianini N. and Shawe-Taylor J. *An introduction to Support Vector Machines.* Cambridge University Press, 2000.

[12] Vladimir Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, 1999.

[13] Cortes Corinna and Vladimir Vapnik. Support vector networks. *Machine Learning*, 20(3):273–297, 1995.

[14] Arne Halaas, Børge Svingen, Magnar Nedland, Pål Sætrom, Ola Jr. Snøve, and Olaf René Birkeland. A recursive MISD architecture for pattern matching. *IEEE Transactions on very large scale integration (VLSI) systems*, 12(7), 2004.

[15] Interagon. IQL documentation download site. http://www.interagon.com/whitepapers.cgi.

[16] Saul Needleman and Christian Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[17] T.F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[18] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J Lipman. A basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[19] W. R. Pearson. Rapid and sensitive sequence comparisons with FASTP and FASTA. *Methods in Enzymology*, 183:63–98, 1990.

[20] 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(22):4673–80, 1994.

[21] Stephen F. Altschul, Thomas L. Madden, Alejandro A. Schaffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and et al. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25:3389–3402, 1997.

[22] J.R. Koza and David Andre. Automatic discovery of protein motifs using genetic programming. *Evolutionary Computation: Theory and Applications*, 1995.

[23] C.J.A. Sigrist, L. Cerutti, N. Hulo, A. Gattiker, L. Falquet, M. Pagni, A. Bairoch, and P Bucher. PROSITE: a documented database using patterns and profiles as motif descriptors. *Briefings in Bioinformatics*, 3:265–274, 2002.

[24] Bjorn Olsson. Using evolutionary algorithms in the design of protein fingerprints. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 1999.

[25] Hu Yuh-Jyh. Biopattern discovery by genetic programming. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, 1998.

[26] Brian J. Ross. The evolution of stochastic regular motifs for protein sequences. *New Generation Computing*, 20(2), 2002.

[27] Rolv Seehus, Amund Tveit, and Ole Edsberg. Discovering biological motifs with genetic programming. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, 2005.

[28] C. Liao and W. C. Noble. Combining pairwise sequence similarity and support vector machines for remote protein homology detection. In *Proceedings of the sixth annual international conference on research in computational molecular biology*, 2002.

[29] M. Gribskov, R. Lüthy, and D. Eisenberg. Profile analysis. *Methods in Enzymology*, 183:146–159, 1990.

[30] A. Krogh, M. Brown, I. Mian, K. Sjolander, and D. Haussler. Hidden markov models in computational biology: Applications to protein modelling. *Journal of Molecular Biology*, 235:1501–1531, 1994.

[31] K. Karplus, C. Barrett, and R. Hughey. Hidden markov models for detecting remote protein homologies. *Bioinformatics*, 14(10):846–856, 1998.

[32] T. Jaakkola, M. Diekhans, and D. Haussler. Using the fisher kernel method to detect remote protein homologies. *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 149–158, 1999.

[33] B. Logan, P. Moreno, B. Suzek, Z. Weng, and S. Kasif. A study of remote homology detection. Technical report, Compaq Cambridge Research Laboratory, 2001.

[34] Asa Ben-Hur and Douglas Brutlag. Remote homology detection: a motif based approach. *Bioinformatics*, 19:26i–33, 2003.

[35] Su Qiaojuan, Lu Lin, and Douglas Brutlag. eBLOCKS. www.motif.stanford.edu/eblocks.

[36] Craig Nevill-Manning, Thomas Brutlag, and Douglas Brutlag. Highly specific protein sequence motifs for genome analysis. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 95, 1998.

[37] C. Leslie, E. Eksin, and W. S. Noble. The spectrum kernel: A string kernel for SVM protein classification. In R. B. Altman, A. K. Dunker,

L. Hunter, K. Lauerdale, and T. E. Klein, editors, *Proceedings of the Pacific Symposium on Biocomputing*, pages 564–575, 2002.

[38] C. Leslie, E. Eleazar, A. Cohen, J. Weston, and W. Stafford Noble. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20:467–476, 2004.

[39] R. Kuang, Eugene Ie, Ke Wang, Kai Wang, Mahira Siddiqi, Yoav Freund, and Christina Leslie. Profile based string kernels for remote homology detection and motif extraction. *Computational Systems Bioinformatics*, 2005.

[40] C. Leslie, E. Eskin, J. Weston, and W. S. Noble. Mismatch string kernels for SVM protein classification. *Advances in Neural Information Processing Systems*, 15:1417–1424, 2003.

[41] Thomas Plötz and Gernot A. Fink. Robust remote homology detection by feature based profile hidden markov models. *Statistical Applications in Genetics and Molecular Biology*, 4(1), 2005.

[42] Yuna Hou, Wynne Hsu, Li Mong Lee, and Christopher Bystroff. Efficient remote homology detection using local structure. *Bioinformatics*, 17(19), 2003.

[43] Yuna Hou, Wynne Hsu, Li Mong Lee, and Christopher Bystroff. Remote homolog detection using local sequence-structure correlations. *Proteins*, 57:518–530, 2004.

[44] C. Bystroff, V. Thorsson, and D. Baker. Hmmstr: a hidden markov model for local sequence-structure correlations in proteins. *Journal of Molecular Biology*, 301(1), 2000.

[45] Jason Weston, Christina Leslie, Eugene Ie, Dengyong Zhou, Andre Elisseeff, and William Stafford Noble. Semi-supervised protein classification using cluster kernels. *Bioinformatics*, 21(15), 2005.

[46] H. Saigo and et al. Protein remote homology detection using string alignment kernels. *Bioinformatics*, 20:1682–1689, 2004.

[47] Huzefa Rangwala and George Karypis. Profile-based direct kernels for remote homology detection and fold recognition. *Bioinformatics*, 21(23):4239–4247, 2005.

[48] 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.

[49] S.E. Brenner, P. Koehl, and M. Levitt. The ASTRAL compendium for sequence and structure analysis. *Nucleic Acids Research*, 28:254–256, 2000.

[50] Q. Su, S. Saxonov, and D. Brutlag. eblocks: An automated database of protein conserved regions maximizing sensitivity and specificity. http://motif.stanford.edu/eblocks.

[51] B. Boeckmann, A. Bairoch, R. Apweiler, M.-C. Blatter, A. Estreicher, E. Gasteiger, M.J. Martin, K. Michoud, C. O'Donovan, I. Phan, S. Pilbout, and M Schneider. The swiss-prot protein knowledgebase and its supplement trembl in 2003. *Nucleic Acids Res.*, 31:365–370, 2003.

[52] Pål Sætrom. Predicting the efficacy of short oligonucleotides in antisense and rnai experiments with boosted genetic programming. *Bioinformatics*, 20(17):3055–3063, 2004.

[53] Arne Johan Hestnes. Protein family prediction. Master's thesis, Norwegian University of Science and Technology, 2005.

[54] D.A Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, and D. L. Wheeler. Genbank. *Nucleic Acids Research*, 34, 2006.

[55] P. Pavlidis, I. Wapinski, and W. S. Noble. Support vector machine classification on the web. *Bioinformatics*, 20:586–587, 2004.

[56] J. Egan. Signal detection theory and ROC analysis. *Series in Cognition and Perception*, 1975.

[57] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics*, 1:80–83, 1945.

[58] Qi wen Dong, Xiao long Wang, and Lei Lin. Application of latent semantic analysis to protein remote homology detection. *Bioinformatics*, 22(3):285–290, 2006.

[59] Nicola J. Mulder, Rolf Apweiler, Teresa K. Attwood, Amos Bairoch, Alex Bateman, David Binns, Paul Bradley, Peer Bork, Phillip Bucher, Lorenzo Cerutti, Richard Copley, Emmanuel Courcelle, Ujjwal Das, Richard Durbin, Wolfgang Fleischmann, Julian Gough, Daniel Haft, Nicola Harte, Nicolas Hulo, Daniel Kahn, Alexander Kanapin, Maria Krestyaninova, David Lonsdale, Rodrigo Lopez, Ivica Letunic, Martin Madera, John Maslen, Jennifer McDowall, Alex Mitchell, Anastasia N. Nikolskaya, Sandra Orchard, Marco Pagni, Chris P. Ponting, Emmanuel Quevillon, Jeremy Selengut, Christian J. A. Sigrist, Ville Silventoinen, David J. Studholme, Robert Vaughan, and Cathy H. Wu. InterPro, progress and status in 2005. *Nucl. Acids Res.*, 33(Database Issue):D201–205, 2005.

[60] Scott Deerwester, Susan T. Dumais, George Furnas, Thomas Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[61] T.K. Landauer, P.W. Foltz, and D. Laham. Introduction to latent semantic analysis. *Discourse Processes*, 25:259–284, 1998.

[62] Latent semantic analysis. http://en.wikipedia.org/wiki/Latent$_s emantic_a nalysis$.

[63] K.Beck. *Test-Driven Development by Example*. Addison Wesley, 2003.

[64] K. Beck. Simple smalltalk testing: With patterns. Technical report, Smalltalk Report, 1994.

Appendices

Appendix A

Additional results

This appendix gives additional results and illustrations not included in the Result and Discussion chapters. The first section discusses the use of Latent semantic analysis on the GPkernel and the Spectrum kernel to improve classification performance. The second section gives the ROC-50 results for the Spectrum kernel with different k-mer lengths. The third section gives the ROC-50 scores on the superfamily benchmark for all test families and for a selection of the methods while the fourth section gives the ROC-50 scores for all test superfamilies in the fold benchmark for the same set of methods.

A.1 Latent semantic analysis for noise removal

The motifs produced for the GPkernel have a variable quality and the motif matrix for a classifier will have several identical or equal motifs, due to the nature of the genetic programming process. Based on this observation, we propose to use a technique for feature extraction and noise removal to get more efficient feature representations of the proteins. This section discusses the use of Latent semantic analysis [60] on the GPkernel and the Spectrum kernel.

A.1.1 The technique of Latent semantic analysis

Latent semantic analysis [60] is a technique in natural language processing that has recently been used with success in remote homology detection [58]. The basic idea is, given a matrix describing the occurrences of patterns in the training set of proteins, to find a low-rank approximation to the matrix by using singular value decomposition and removing the smallest singular values in the diagonal matrix and the corresponding rows and columns in the two other matrices that result from the decomposition. This condenses

correlated features, resulting in a smaller feature space with less noise than the original vector representation. The latent semantic analysis process is only performed on the training data; the test data are thereafter converted to the smaller feature space.

Using our motif matrix $A(M, N)$ giving the number of motif matches for motif m in protein sequence n, we split this into a training matrix $W(M, I)$ and a test matrix $T(M, J)$ where the training matrix has all instances of the training set and the test matrix has all instances of the test set. The training matrix is then decomposed by singular value decomposition into three matrices:

$$W = U \times S \times V^T$$

U is left singular matrix having M rows and K columns, S is the $K \times K$ diagonal matrix and V^T is the right singular matrix having K rows and I columns. Only a given number of the largest values from the diagonal matrix and the corresponding columns of matrix U and rows of matrix V^T are then kept. The rest of S, U and V^T is ignored.

A new training matrix is then computed from the column vectors of the reduced matrix V^T, which are equal to $S^{-1} \times U^T \times W_i$, where W_i denotes column i in the training matrix. The vectors of the new training matrix are of a smaller dimension than the original vectors of the training matrix. To convert the test vectors to this smaller feature space, each column vector T_j is multiplied with $S^{-1} \times U^T$.

It has been reported that keeping only 200-300 features is typical and gives good results [58]. Our motif matrix for the GPkernel has 3350 motifs. Computing a new feature matrix with only 300 features therefore means to reduce the number of feature space dimensions to one tenth of the original. For comparison, LSA is also performed on a Spectrum kernel [37] with k-mer length 3.

A.1.2 Performance for GPkernel and Spectrum is reduced with LSA

Figures A.1 and A.2 show the result of using the process of latent semantic analysis (LSA) on the motif matrix of the GPkernel. Figure A.2 is a family by family comparison of the ROC-50 scores for the GPkernel and for the GPkernel with LSA. All LSA results are based on reducing the number of features to 300. A few test sets are better classified with LSA, but overall the GPkernel is significantly better without LSA (p-value $1.07 \cdot 10^{-10}$ on a Wilcoxon signed rank test).

Figures A.3 and A.4 show the results of using LSA on the Spectrum kernel with k-mer length 3. The results show that also the Spectrum kernel is significantly better without LSA (p-value $4.98 \cdot 10^{-10}$).

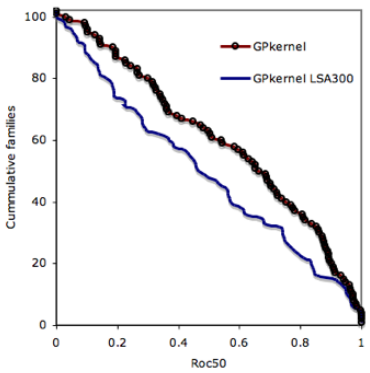

Figure A.1: ROC-50 results on superfamily benchmark for GPkernel with Latent semantic analysis.

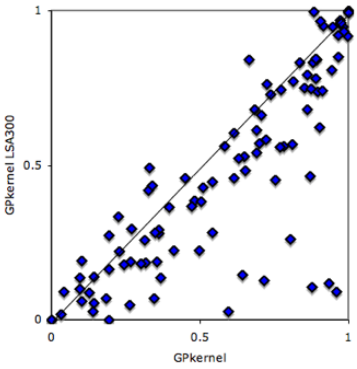

Figure A.2: A family by family comparison of the ROC-50 results on superfamily benchmark for GPkernel and GPkernel with Latent semantic analysis.

A.1.3 Initial experiments with LSA are inconclusive

Because genetic programming is a stochastic process, the GP algorithm will sometimes produce motifs of a quality that are not helpful to the GPkernel. The fact that many of the GP motifs may be redundant or be inferior compared to the better motifs motivated the use of latent semantic analysis (LSA) on the GPkernel motif matrix to remove noise and improve classification accuracy. But as seen in section A.1.2, the original GPkernel is significantly better than the alternative GPkernel after using LSA. It might be that the GP motif matrix is too dense or that the GP motifs do not correlate into meaningful concepts suitable for discovery of the LSA process. Instead of removing noise, LSA might be removing too much important information from the kernel.

The number of dimensions retained in LSA is an empirical issue. Our choice of 300 features is based on the recommendations in the original article of LSA in protein remote homology detection [58]. Instead of trying to optimise this parameter, we have investigated the results of performing LSA on the Spectrum kernel. Our results on the Spectrum kernel do not concede with previous findings, where LSA improved average ROC score by almost 9% on a smaller superfamily benchmark based on the 1.53 version of the

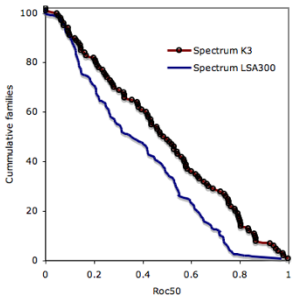

Figure A.3: ROC-50 results on superfamily benchmark for Spectrum kernel with Latent semantic analysis.

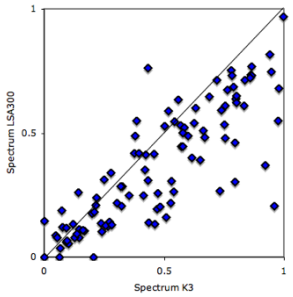

Figure A.4: A family by family comparison of the ROC-50 results on superfamily benchmark for Spectrum kernel and Spectrum with Latent semantic analysis.

SCOP database [58]. On our bigger superfamily benchmark, the average ROC score is almost 7% higher for the Spectrum kernel without LSA than for Spectrum with LSA and the differences are significant on both ROC scores ($1.37 \cdot 10^{-12}$) and ROC-50 scores (p-value $8.85 \cdot 10^{-11}$).

It should be noted that the original article on LSA in protein remote homology detection [58] is poorly written and has unfortunate many errors. Our setup for the computation of the Spectrum kernel is the same as in [58], but since following the article's description of the LSA method did not yield good results and since we were unable to get in contact with the corresponding author of the article, we based the setup for the LSA step on two other sources [61] [62]. This yielded the results reported in the previous section. It has later come to our attention that a third alternative description of LSA exist [60] and we therefore conclude that the results reported here should be viewed as preliminary results and that additional experiments are needed to reach any conclusions.

Hypothetically, it could be interesting to see the effect of LSA on the eMOTIF kernel. The eMOTIF kernel is a much more sparse kernel than the GPkernel and it is possible that this would be favourable for LSA. Unfortunately, performing LSA is quite memory demanding. Our current implementation requires the explicit storage of the motif occurrence matrix. While this is not strictly necessary, the matrices resulting from the decomposition would have to be stored explicitly. With 500000 eMOTIFs, the U matrix would then be of dimensions $500000 * K$, where K is the rank of the occurrence matrix. In this case the rank would not be more than the number of sequences, approximately 4000. With single precision floating point numbers

(8 bytes), the U matrix alone could in theory require $(500000 * 4000 * 8B)$ = 16GB of storage space before feature reduction. Because different matrices must be made for each classifier, performing LSA on the eMOTIF data would therefore not be practical.

A.2 ROC-50 results for Spectrum kernel

Figures A.5 and A.6 shows how different k-mer lengths affect the performance of the spectrum kernel. The figures show that k-mers of 3 characters gives the best results for the Spectrum kernel. The Mismatch kernel makes it possible to use longer k-mers, giving more relevant features and better results [38]. The potential feature space of the Spectrum kernel decreases rapidly with shorter k-mer lengths and the variance in the value range of each feature increases. This gives better results, and is consistent with what was found in the original article [37].

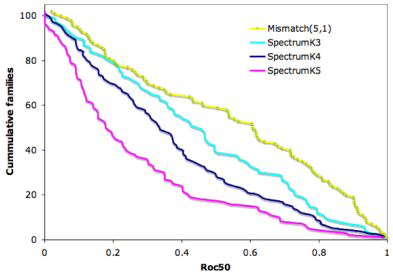

Figure A.5: ROC-50 results on superfamily benchmark for different k-mer lengths.

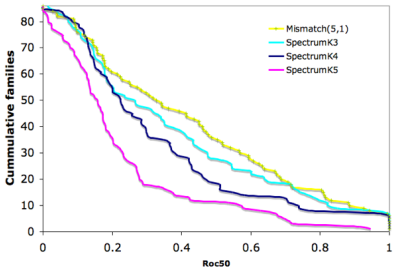

Figure A.6: ROC-50 results on fold benchmark for different k-mer lengths.

A.3 Results for superfamily benchmark

Table A.1 gives the ROC-50 results of a selection of the methods for the superfamily benchmark. PSI-BLAST is based on the average E-value of the profiles made from the NCBI non-redundant database. SVM-Pairwise is made from pairwise BLAST alignments.

Table A.1: ROC-50 results for all test sets on superfamily benchmark

Family	GPKernel	GPboost	eMOTIF	Mismatch	SpectrumK3	SVM-Pairwise	PSI-Blast
a.1.1.2	0.270	0.000	0.785	0.284	0.253	0.365	0.888
a.1.1.3	0.193	0.000	0.058	0.152	0.196	0.058	0.000
a.3.1.1	0.999	0.831	0.933	0.771	0.404	0.922	1.000
a.4.1.1	0.942	0.056	0.287	0.516	0.477	0.386	0.000
a.22.1.1	0.932	0.086	1.000	0.832	0.744	0.953	1.000
a.22.1.3	0.595	0.000	0.818	0.478	0.373	0.974	0.825
a.25.1.1	0.482	0.047	0.148	0.133	0.130	0.300	0.684
a.25.1.2	0.127	0.473	0.116	0.232	0.128	0.203	0.113
a.26.1.1	0.877	0.535	0.218	0.697	0.398	0.374	0.000
a.26.1.2	0.871	0.000	0.403	0.748	0.513	0.352	0.050
a.26.1.3	0.685	0.315	0.337	0.724	0.453	0.448	0.059
a.39.1.2	0.971	0.377	0.990	0.929	0.650	0.999	1.000
a.39.1.5	0.851	0.756	0.960	0.901	0.718	0.976	1.000
a.39.1.8	0.915	0.960	1.000	0.995	0.829	1.000	1.000
a.138.1.1	1.000	1.000	1.000	1.000	0.997	1.000	0.292
a.138.1.3	0.998	0.912	1.000	0.905	0.837	0.983	0.784
b.1.1.1	0.691	0.287	0.993	0.733	0.568	0.986	0.976
b.1.1.2	0.973	0.547	0.923	0.932	0.937	0.968	0.949
b.1.1.3	0.650	0.056	0.909	0.790	0.737	0.846	0.732
b.1.1.4	0.892	0.112	0.966	0.651	0.601	0.891	0.879
b.1.18.2	0.043	0.000	0.180	0.048	0.069	0.164	0.378
b.6.1.1	0.317	0.000	0.889	0.465	0.504	0.708	0.457
b.6.1.3	0.471	0.112	0.269	0.267	0.229	0.572	0.844
b.121.4.1	0.325	0.567	0.460	0.204	0.284	0.260	0.674
b.121.4.7	0.879	0.236	0.302	0.808	0.834	0.112	0.000
b.29.1.1	0.652	0.187	0.125	0.516	0.375	0.284	1.000
b.29.1.2	0.614	0.055	0.057	0.193	0.334	0.417	0.561
b.29.1.3	0.000	0.000	0.194	0.097	0.075	0.107	0.000
b.29.1.11	0.782	0.000	0.369	0.662	0.491	0.333	0.530
b.40.2.1	0.194	0.000	0.053	0.083	0.157	0.052	0.047
b.40.2.2	0.184	0.000	0.069	0.215	0.157	0.117	0.061
b.40.4.3	0.395	0.068	0.177	0.151	0.065	0.085	0.139
b.40.4.5	0.540	0.057	0.172	0.438	0.346	0.346	0.000
b.47.1.1	0.725	0.650	0.784	0.889	0.757	0.913	0.866
b.47.1.2	0.641	0.879	1.000	0.988	0.774	0.989	1.000
b.47.1.3	0.665	0.547	0.901	0.928	0.453	0.960	0.636
b.50.1.1	0.225	0.000	0.924	0.569	0.784	0.436	0.000
b.50.1.2	0.031	0.300	0.826	0.149	0.297	0.285	0.048

Continued on next page

Table A.1 – continued from previous page

Family	GPKernel	GPboost	eMOTIF	Mismatch	SpectrumK3	SVM-Pairwise	PSI-Blast
b.55.1.1	0.266	0.000	0.183	0.196	0.055	0.299	0.944
b.60.1.1	0.510	0.000	0.240	0.210	0.229	0.420	0.832
b.60.1.2	0.498	0.000	0.065	0.277	0.231	0.344	0.000
b.82.1.2	0.361	0.071	0.069	0.475	0.470	0.599	0.932
c.1.2.4	0.542	0.358	0.420	0.785	0.529	0.800	0.736
c.1.8.1	0.960	0.937	0.219	0.726	0.861	0.431	0.775
c.1.8.3	0.700	0.740	0.222	0.619	0.633	0.346	0.332
c.1.8.4	0.946	0.911	0.789	0.905	0.955	0.706	1.000
c.1.8.5	0.774	0.710	0.261	0.627	0.798	0.328	0.122
c.1.10.1	0.411	0.126	0.238	0.558	0.400	0.358	0.126
c.2.1.1	0.814	0.448	0.833	0.626	0.597	0.583	0.899
c.2.1.2	0.313	0.432	0.437	0.404	0.279	0.348	0.707
c.2.1.3	0.144	0.061	0.473	0.098	0.126	0.439	0.485
c.2.1.4	0.714	0.499	0.857	0.615	0.507	0.900	0.855
c.2.1.5	0.894	0.282	0.868	0.555	0.369	0.531	0.728
c.2.1.6	0.737	0.431	0.787	0.318	0.439	0.789	0.853
c.2.1.7	0.328	0.195	0.330	0.383	0.340	0.506	0.651
c.3.1.2	0.810	0.995	0.993	0.860	0.486	0.763	0.944
c.3.1.5	0.339	0.281	0.814	0.758	0.376	0.802	0.937
c.26.1.1	0.613	0.735	0.401	0.279	0.264	0.323	0.141
c.26.1.3	0.502	0.356	0.593	0.256	0.252	0.648	1.000
c.37.1.1	0.704	0.790	0.945	0.711	0.389	0.862	0.850
c.37.1.8	0.448	0.404	0.994	0.780	0.608	0.730	0.349
c.37.1.9	0.754	0.964	0.892	0.945	0.719	0.714	0.290
c.37.1.10	0.769	0.930	0.738	0.897	0.724	0.820	0.775
c.37.1.11	0.889	0.938	0.992	0.868	0.750	0.768	0.570
c.37.1.12	0.873	0.981	0.999	0.984	0.817	0.999	0.896
c.37.1.19	0.689	0.348	0.623	0.617	0.708	0.467	0.702
c.37.1.20	0.836	0.851	0.856	0.904	0.716	0.878	0.765
c.45.1.2	0.142	0.458	0.916	0.565	0.072	1.000	1.000
c.47.1.1	0.722	0.023	0.807	0.308	0.305	0.910	0.963
c.47.1.5	0.095	0.000	0.130	0.109	0.142	0.209	0.204
c.47.1.10	0.243	0.108	0.323	0.140	0.198	0.444	0.362
c.55.1.1	0.362	0.026	0.352	0.202	0.098	0.209	0.170
c.55.1.3	0.299	0.071	0.343	0.162	0.000	0.104	0.189
c.55.3.1	0.264	0.067	0.070	0.281	0.066	0.048	0.000
c.55.3.5	0.349	0.221	0.118	0.303	0.245	0.142	0.115
c.56.5.4	0.229	0.076	0.252	0.204	0.000	0.309	0.744
c.67.1.1	0.582	0.626	0.595	0.626	0.489	0.639	1.000

Continued on next page

Table A.1 – continued from previous page

Family	GPKernel	GPboost	eMOTIF	Mismatch	SpectrumK3	SVM-Pairwise	PSI-Blast
c.67.1.3	0.859	0.812	0.456	0.834	0.769	0.832	1.000
c.67.1.4	0.910	0.906	0.621	0.837	0.822	0.746	1.000
c.94.1.1	0.368	0.130	0.112	0.278	0.238	0.036	0.184
c.94.1.2	0.102	0.253	0.133	0.413	0.485	0.000	0.265
d.2.1.2	0.141	0.000	0.247	0.040	0.000	0.110	0.000
d.2.1.3	0.805	0.000	0.045	0.295	0.209	0.000	0.000
d.3.1.1	0.102	0.159	0.046	0.075	0.047	0.349	0.074
d.15.1.1	0.344	0.000	0.174	0.216	0.411	0.202	0.095
d.15.4.1	0.964	0.851	0.889	0.738	0.792	0.771	1.000
d.15.4.2	0.891	0.565	0.890	0.558	0.522	0.603	0.785
d.32.1.3	0.631	0.175	0.645	0.752	0.604	0.892	1.000
d.81.1.1	0.000	0.000	0.000	0.107	0.127	0.100	0.000
d.92.1.11	0.906	0.885	0.888	0.922	0.732	1.000	1.000
d.108.1.1	0.193	0.089	0.413	0.139	0.174	0.122	0.850
d.153.1.4	0.093	0.000	0.120	0.159	0.093	0.111	0.000
d.169.1.1	0.354	0.000	0.395	0.285	0.200	0.323	0.096
g.3.6.1	0.998	0.954	0.906	0.948	0.978	0.974	0.000
g.3.6.2	0.999	0.947	0.811	0.848	0.941	0.842	0.000
g.3.7.1	0.981	0.964	0.976	0.947	0.893	0.966	0.000
g.3.7.2	0.966	0.728	0.956	0.951	0.955	0.995	0.000
g.3.11.1	0.862	0.405	0.960	0.627	0.484	0.768	0.335
g.14.1.1	0.975	0.738	0.729	0.882	0.574	0.811	0.138
g.14.1.2	0.882	0.215	0.333	0.661	0.372	0.742	0.000
g.39.1.2	0.902	0.961	0.817	0.863	0.632	0.783	0.000
g.39.1.3	0.985	0.061	0.976	0.842	0.702	0.923	0.303

A.4 Results for fold benchmark

Table A.2 gives the ROC-50 results of a selection of the methods for the fold benchmark. PSI-BLAST is based on the average E-value of the profiles made from the NCBI non-redundant database. SVM-Pairwise is made from pairwise BLAST alignments.

Table A.2: ROC-50 results for all test sets on fold benchmark

Superfamily	GPKernel	GPboost	eMOTIF	Mismatch	SpectrumK3	SVM-Pairwise	PSI-Blast
a.2.11	0.322	0.000	0.039	0.214	0.203	0.224	0.000
a.4.1	0.782	0.006	0.239	0.699	0.640	0.458	0.130
a.4.5	0.487	0.029	0.192	0.440	0.422	0.265	0.106
a.4.6	0.799	0.220	0.520	0.710	0.789	0.573	0.380
a.5.2	0.597	0.000	0.265	0.388	0.244	0.307	0.000
a.7.1	0.760	0.116	0.472	0.585	0.668	0.512	0.000
a.60.1	0.452	0.000	0.071	0.192	0.168	0.158	0.000
a.102.1	0.927	0.975	0.470	0.822	0.848	0.556	0.329
a.102.4	0.797	0.811	0.264	0.727	0.755	0.648	0.837
a.118.1	0.393	0.646	0.106	0.407	0.259	0.139	0.475
a.118.8	0.730	0.394	0.301	0.826	0.742	0.527	0.801
b.1.1	0.645	0.135	0.463	0.608	0.546	0.406	0.114
b.1.18	0.552	0.104	0.283	0.525	0.547	0.405	0.132
b.1.2	0.671	0.083	0.261	0.494	0.469	0.291	0.480
b.1.6	0.788	0.119	0.292	0.545	0.436	0.405	0.068
b.1.8	0.504	0.000	0.283	0.188	0.220	0.255	0.000
b.2.2	0.444	0.061	0.063	0.488	0.475	0.215	0.000
b.2.3	0.744	0.152	0.171	0.554	0.343	0.254	0.075
b.2.5	0.305	0.000	0.162	0.065	0.137	0.000	0.000
b.121.4	0.736	0.678	0.126	0.573	0.529	0.125	0.000
b.121.5	0.943	0.856	0.282	0.941	0.868	0.166	0.000
b.34.2	0.520	0.000	0.083	0.196	0.153	0.210	0.000
b.34.5	0.508	0.000	0.344	0.221	0.116	0.183	0.000
b.40.2	0.057	0.000	0.048	0.106	0.065	0.076	0.079
b.40.4	0.401	0.018	0.211	0.287	0.197	0.200	0.000
b.40.6	0.512	0.073	0.266	0.602	0.367	0.255	0.000
b.42.1	0.471	0.000	0.000	0.500	0.444	0.559	0.724
b.42.2	0.260	0.000	0.039	0.139	0.191	0.166	0.000
b.43.4	0.121	0.054	0.057	0.267	0.157	0.033	0.066
b.43.3	0.392	0.076	0.122	0.508	0.376	0.265	0.000
b.68.1	0.903	0.945	0.128	0.720	0.832	0.248	0.111
b.69.4	0.799	0.772	0.731	0.669	0.624	0.468	1.000
b.80.1	0.885	0.958	0.249	0.679	0.493	0.170	0.000
b.82.1	0.315	0.099	0.127	0.188	0.219	0.202	0.577
b.82.2	0.367	0.643	0.167	0.164	0.223	0.321	0.267
b.82.3	0.192	0.000	0.128	0.178	0.372	0.213	0.236
b.84.1	0.576	0.000	0.215	0.625	0.412	0.825	0.721
c.1.1	0.204	0.161	0.154	0.485	0.175	0.595	0.260

Continued on next page

Table A.2 – continued from previous page

Superfamily	GPKernel	GPboost	eMOTIF	Mismatch	SpectrumK3	SVM-Pairwise	PSI-Blast
c.1.2	0.396	0.179	0.624	0.729	0.567	0.793	0.929
c.1.4	0.666	0.609	0.657	0.834	0.727	0.726	0.784
c.1.7	0.376	0.301	0.438	0.489	0.430	0.475	0.318
c.1.8	0.201	0.483	0.266	0.346	0.309	0.143	0.283
c.1.9	0.764	0.851	0.659	0.731	0.624	0.262	0.294
c.1.10	0.685	0.600	0.477	0.826	0.787	0.470	0.439
c.1.11	0.716	0.478	0.281	0.644	0.501	0.258	0.474
c.1.12	0.852	0.714	0.506	0.847	0.768	0.352	0.744
c.1.15	0.967	0.931	0.670	0.949	0.956	0.476	0.544
c.23.1	0.240	0.000	0.110	0.313	0.357	0.198	0.609
c.23.5	0.258	0.140	0.070	0.358	0.422	0.168	0.128
c.23.12	0.499	0.045	0.078	0.472	0.202	0.277	0.000
c.23.16	0.386	0.247	0.209	0.363	0.407	0.265	0.180
c.26.1	0.305	0.472	0.112	0.324	0.216	0.178	0.000
c.26.2	0.341	0.099	0.154	0.437	0.161	0.142	0.093
c.55.1	0.295	0.240	0.126	0.134	0.115	0.081	0.000
c.55.3	0.339	0.425	0.240	0.155	0.277	0.276	0.000
c.56.2	0.858	0.890	0.334	0.625	0.494	0.267	0.000
c.56.5	0.293	0.234	0.076	0.116	0.196	0.030	0.020
d.15.1	0.491	0.000	0.170	0.281	0.183	0.237	0.056
d.15.2	0.338	0.000	0.324	0.140	0.083	0.313	0.000
d.15.4	0.362	0.000	0.149	0.105	0.159	0.112	0.140
d.15.6	0.358	0.000	0.221	0.248	0.310	0.158	0.000
d.15.7	0.041	0.000	0.445	0.657	0.504	0.638	0.084
d.17.2	0.141	0.000	0.137	0.229	0.187	0.145	0.000
d.17.4	0.094	0.000	0.099	0.050	0.146	0.106	0.069
d.26.1	0.363	0.021	0.480	0.338	0.224	0.181	0.000
d.26.3	0.395	0.000	0.164	0.211	0.129	0.161	0.000
d.58.1	0.123	0.043	0.000	0.275	0.128	0.145	0.080
d.58.3	0.351	0.000	0.051	0.263	0.078	0.458	0.066
d.58.5	0.324	0.000	0.289	0.436	0.296	0.137	0.103
d.58.7	0.172	0.000	0.080	0.104	0.106	0.218	0.059
d.58.17	0.129	0.000	0.041	0.251	0.127	0.165	0.077
d.58.26	0.000	0.133	0.215	0.070	0.082	0.391	0.169
d.79.1	0.415	0.000	0.081	0.198	0.085	0.169	0.000
d.110.3	0.293	0.000	0.343	0.126	0.362	0.420	0.000
d.129.1	0.109	0.000	0.209	0.000	0.052	0.060	0.000
d.129.3	0.050	0.000	0.000	0.000	0.052	0.102	0.000
f.1.4	0.365	0.178	0.053	0.182	0.206	0.192	0.000

Continued on next page

Table A.2 – continued from previous page

Superfamily	GPKernel	GPboost	eMOTIF	Mismatch	SpectrumK3	SVM-Pairwise	PSI-Blast
g.3.1	1.000	1.000	1.000	1.000	1.000	1.000	0.000
g.3.2	1.000	0.992	1.000	1.000	1.000	1.000	0.000
g.3.6	1.000	0.988	0.999	0.993	0.998	0.983	0.000
g.3.7	1.000	1.000	0.997	1.000	1.000	1.000	0.000
g.3.9	1.000	0.913	0.923	1.000	1.000	0.992	0.000
g.3.11	1.000	0.981	0.997	1.000	1.000	1.000	0.000
g.3.13	1.000	1.000	1.000	1.000	1.000	1.000	0.000
g.41.3	0.977	0.595	0.567	0.914	0.686	0.766	0.000
g.41.5	0.995	0.686	0.941	0.898	0.812	0.941	0.063

Appendix B

Programming methodology

This appendix explains the methodology used for a substantial part of the programming done during the work on this thesis.

B.1 Test-driven development

Test-driven development (TDD) [63] is a methodology meant to improve the process of software development and in the end, the quality of software, by providing the programmer with rapid feedback through the development process. The idea is to repeatedly first write a simple test case and then write the code necessary to pass the test. The development process is aided by some form of automatic unit testing framework so it is possible to easily run a series of tests in each cycle. A test driven development cycle consists of the following steps:

1. **Make a test -** The first step of TDD is to write a test that uses the functionality that is to be implemented in the next steps.

2. **Run all tests -** By running all tests, the programmer verifies that the new test actually tests something. If the new test does not fail, this informs the programmer that the desired functionality is somehow already implemented, which probably means there is an error somewhere in the code or in the programmer's understanding of the code.

3. **Implement some code to pass the test -** The main point of the next step is to write just enough code that the programmer thinks is necessary to pass the test. The programmer is for the moment allowed to "fake" a solution, by for example hard-coding return values so the test will run and pass.

4. **Run all tests again -** All the tests should now pass.

5. **Refactor code -** The final step is to refactor and clean up code so that the program really implements the desired functionality. Tests are run when necessary so that the programmer is confident that the results of the refactoring do not make the code fail.

This cycle is repeated continuously. Writing a test before implementing functionality ensures that the programmer has a clear idea of what the next action he needs to take is, which is beneficial to the development speed. It is not necessary to write tests for every line of code, the amount of tests written in each cycle depends on the complexity of the problem being solved and the competence of the programmer.

The result of TDD is increased confidence in the correctness of the code. TDD can also improve the design of code since writing a test first forces the programmer to start from the interface perspective of the program, by concentrating on how the desired functionality will be used. The development process also ensures that a substantial part of the code has unit tests, which means that problems and errors in the code will be discovered in an early phase of development instead of in later stages through tedious debugging.

In this thesis, the xUnit framework [64] has been used to automate unit testing in C++, Java and Python. It is especially valuable to use unit testing to test program behaviour with boundary values in input. For example, when making the program to calculate ROC-50 scores, testing what the ROC-50 score is when the 50 highest discriminators are all equal, clarifies both the code and provides interesting reasoning regarding the nature of the ROC curve.

Appendix C

Article

The following pages of this appendix give the article that currently is in the review process at BMC Bioinformatics.

Motif kernel generated by genetic programming improves remote homology and fold detection

Tony Håndstad[1] , Arne Johan Husebø Hestnes[1] and Pål Sætrom*[1,2]

[1]Department of Computer and Information Science, Norwegian University of Science and Technology, NO-7052,Trondheim, Norway
[2]Interagon AS, Laboratoriesenteret, NO-7006 Trondheim, Norway

Email: Tony Håndstad - handstad@stud.ntnu.no; Arne Johan Husebø Hestnes - arnejohe@gmail.com; Pål Sætrom*- paal.saetrom@interagon.com;

*Corresponding author

Abstract

Background: Protein remote homology detection is a central problem in computational biology. Most recent methods train support vector machines to discriminate between related and unrelated sequences and these studies have introduced several types of kernels. One successful approach is to base a kernel on shared occurrences of discrete sequence motifs. Still, many protein sequences fail to be classified correctly for a lack of a suitable set of motifs for these sequences.

Results: We introduce the GPkernel, which is a motif kernel based on discrete sequence motifs where the motifs are evolved using genetic programming. All proteins can be grouped according to evolutionary relations and structure, and the method uses this inherent structure to create groups of motifs that discriminate between different families of evolutionary origin. When tested on two SCOP benchmarks, the superfamily and fold recognition problems, the GPkernel gives significantly better results compared to related methods of remote homology detection.

Conclusions: The GPkernel gives particularly good results on the more difficult fold recognition problem compared to the other methods. This is mainly because the method creates motifs that not only describe similarities among the related proteins, but also similarities among subgroups of the unrelated proteins. This rich set of motifs gives a better description of the similarities and differences between different folds than do previous motif-based methods.

Background

A huge gap exists between the number of protein sequences and the number of proteins with a known structure and function. The exponential growth in sequence information means that better methods to automatically annotate new sequences are needed. An important problem in computational biology is therefore the detection of subtle sequence similarities, as this might imply that the sequences share a common ancestor — that is, they are remote homologues — suggesting structural and functional similarities. Several good solutions exist when the level of sequence similarity is high, but when the sequences are highly divergent it is still difficult to distinguish remotely homologue sequences from sequences that are similar by chance.

Early solutions to the problem of finding remote homologues, such as the Smith-Waterman algorithm [1] and heuristic alternatives like BLAST [2] and FASTA [3], looked for sequence similarity between pairs of proteins. Later solutions used aggregated statistics of related proteins to generate more complex models that a protein with unknown function could be compared to. These methods, including profiles [2, 4] and hidden Markov models (HMMs) [5–7] used only related sequences for model generation.

The most successful recent methods have been discriminative. Classifiers are trained on both related and unrelated proteins to recognize what distinguishes the related proteins from the unrelated ones. Kernel methods such as the support vector machine [8] have proven to give particularly good results and several different types of kernel functions have been introduced [9–17]. Most of these kernel functions are typically either based on profiles and sequence alignments or based on the occurrences of discrete motifs.

Kernels based on profiles and sequence alignments

The first method that used support vector machines was the Fisher kernel [9]. This method trains profile HMMs on related proteins and produces feature vectors from sequences by aligning them to the HMMs. Another alignment-based kernel is SVM-Pairwise [10], which represents each sequence as a vector of pairwise similarities to all sequences of the training set. The SVM-I-sites method [11] compares sequence profiles to the I-sites library of local structural motifs for feature extraction and this method has also been improved to take into account the order and relationships of the I-site motifs [12].

A relatively simple but efficient kernel is the Mismatch kernel [13] in which the feature space consists of all short subsequences of length k, called k-mers. A k-mer is said to be present in a sequence if the sequence contains a substring that has at most n mismatches to the k-mer. In the profile kernel of Kuang et al. [14], the mismatch kernel is combined with profiles; a k-mer is said to be present in a sequence if the sequence contains a substring that when aligned to the profile gives a score above a given threshold. Later methods, such as the LA-kernel [15] and SVM-SW [16] are also alignment-based, but instead of representing the sequences as a vector of features they calculate the kernels directly by an explicit protein similarity measure. The LA-kernel uses all optimal gapped local alignment scores for all possible subsequences of two sequences, while SVM-SW uses the optimal local alignment that maximizes a direct profile-profile score.

Kernels based on discrete sequence motif content

Motif kernels are based on the idea of using motif content to measure sequence similarity. Protein sequence motifs describe some common sequence pattern that is conserved over greater evolutionary distance than the rest of the sequences. Focusing on sequence motifs therefore means focusing on the most conserved parts of a sequence, where remote homologues are most likely to share similarities.

Although there are many databases of sequence motifs available [18–21], these databases were created in a supervised way to have motifs that characterize different known protein families, domains, or functional sites. Consequently, a motif kernel based on these databases will be biased towards correctly classifying known functions or families. This also makes such motifs kernels inappropriate in benchmark studies. The eMOTIF kernel of Ben-Hur and Brutlag [17] avoids these problems by using motifs extracted with the unsupervised eMOTIF method [22] from the eBLOCKS database [23]. The eMOTIF kernel has good performance when classifying sequences in classes for which several motifs are available, but the performance decreases when related sequences share few or no motifs [14].

An alternative to using motifs from an existing database is to generate the motifs from the available data. We introduce a motif kernel where genetic programming is used to find discriminative sequence patterns matching the positive training set sequences while not matching the negative training set sequences. The motifs are made from a simple regular expression-like grammar and the resulting matches against the data set is used to build feature vectors for a support vector machine.

We benchmark our GPkernel on updated versions of two commonly used benchmarks [16] based on the SCOP database [24] and compare its performance to the eMOTIF, Mismatch, and SVM-Pairwise kernels as well as with the PSI-BLAST method. We find that our method gives significantly better results than all

these. We also find, when comparing the GPkernel to related motif methods, that motifs trained on the different classes of negative sequences are vital for the method's predictive power.

Results and Discussion

Genetic programming for protein motif discovery

There are several methods that use genetic programming (GP) [25] to evolve Prosite motifs from multiple unaligned [26–28] or aligned [29] sequences. Genetic programming have also been used to create stochastic regular expression motifs [30]. We use GP to evolve the discrete sequence motifs that serve as a basis for our methods.

Our GP algorithm is trained on a positive and negative training set and the fitness of a candidate solution is a function of its matches in the two sets. The fitness evaluation is accelerated by special purpose search hardware [31], which reduces the training time.

The hardware supports several regular expression-like operators, but we use only a small subset of these. Our solution language is formally defined elsewhere [32], but is here modified to handle the protein alphabet of amino acids. The language's basic elements are the amino acid symbols and the wildcard operator that matches any amino acid. The pattern

$$GL.A$$

will for example match $GLAA$, $GLCA$, $GLDA$ and so forth. To allow alternative amino acids at specific positions, patterns can be extended with the logical disjunction operator; for example, the following pattern will match $GLAA$ and $GLCA$:

$$GL\,(A|C)\,A$$

Finally, the Hamming distance operator specifies the minimum number of amino acids that must match in the pattern. The pattern

$$\{GLAA : p \geq 3\}$$

will for example match $GLAA$, $GCAA$, and $ELAA$, but not match $GLCC$ and $GCAQ$.

Note that the Hamming distance operator makes it possible to specify that a pattern can have a certain number of mismatches. It is also possible to boost the importance of certain amino acid residues by using the Hamming operator in combination with the disjunction operator. For example, it is possible to double the weight of the Leucine residue in the above example so that a Leucine matching at the given position

will count as two matches:

$$\{G\,(L|L)\,AA : p \geq 4\}$$

This pattern will now only match sequences that contain Leucine and two of the three other amino acids, such as $ALAA$, $GLCA$, and $GLAE$. This position specific weighting of amino acids can be used to define patterns that approximate position weight matrices.

The GPkernel uses diverse motifs

The problem of protein remote homology detection has an inherent structure as all proteins can be grouped according to evolutionary relations and structure. This is the basis for the SCOP hierarchy [24]. Our method uses this inherent structure to create a kernel based on a rich set of motifs that tries to capture information about all related sequences in a particular dataset.

To compute the kernel, we use the GP algorithm to produce motifs from the SCOP training set. The kernel is therefore referred to as the GPkernel. For each superfamily classifier, we make both positive and negative motifs; that is, motifs trained to match the classifier's superfamily as well as motifs trained to match the other superfamilies. Correspondingly, we do the same for the fold benchmark and train motifs for all folds. This is done based on the hypothesis that motifs trained to recognize the different aspects of the negative data will increase the discriminative power of the GPkernel.

The basic positive training sets for GP include all members of a superfamily or fold, except for the sequences forming the positive test set. This means that all the motifs will have to cover a large taxonomic distance. To narrow the structural range each motif has to cover, we also split the positive training set into subsets, as shown and explained in Figure 1. For the superfamily benchmark, the subsets exclude one family from the superfamily sequences. For each such subset, we make ten motifs. We also make ten motifs trained on all the sequences of the superfamily. As a consequence, the number of motifs produced for each superfamily will be ten times the number of families in the superfamily. In total, this produces 3350 motifs for every classifier in the superfamily benchmark.

For the fold benchmark, it is neither practical nor beneficial to generate motifs for all subsets excluding one superfamily because of the very high number of superfamilies in each fold. Because we would like to base our kernel on approximately the same number of motifs as for the superfamily benchmark, we adopt a slightly different solution. The sequences of a fold are grouped into superfamilies and ten sets are made for each fold that each exclude one tenth of the sequences. This gives 3300 motifs for the fold benchmark.

For both benchmarks, we run GP with a population of 100 candidate solutions for 50 generations. The

resulting motifs are matched against all the sequences to produce a matrix of binary feature vectors. This matrix contains a 1 at position (i, j) if sequence i contains motif j and a 0 otherwise. The Gram matrix is then produced by taking the dot product between the vectors of the sequences; see Methods for additional details.

Boosted classifiers and an extended eMOTIF kernel

The GPkernel, uses motifs made from genetic programming as a basis for a kernel for a support vector machine. We also propose another method in which we use the GPboost program [32] to build boosted classifiers [33]. Each such classifier is based on 100 weighted sub-motifs where each sub-motif is made by running genetic programming on the SCOP training sets with a population of 100 candidate motifs for 100 generations. The prediction of a boosted classifier is a sum of the predictions from the weighted sub-motifs. In addition, 10 boosted classifiers are made and combined so that the final prediction for a new sequence will be the average of 10 boosted classifiers. The setup is explained in Figure 2.
The eMOTIF kernel has shown good performance on protein families that share many eMOTIFs, but the performance decreases for families that are not covered well by the eMOTIFs. We propose to extend the eMOTIF kernel with an additional small set of GP motifs in hope that this will give a better performance when classifying the sequences that share fewer eMOTIFs. The extended eMOTIF kernel, called GPextended, is made from an eMOTIF kernel with additional GP motifs trained to target the positive training set. The motifs are made in exactly the same way as for the GPkernel, but only the positive motifs — the motifs trained on the subsets of a given classifier's training set — are added to an eMOTIF kernel to create the GPextended kernel.

The GPkernel performs significantly better than the other motif-based methods

Figure 3 shows the performance of the GPkernel, GPboost classifier, the eMOTIF kernel, and the GPextended kernel on the superfamily benchmark. The GPkernel has the best overall performance in terms of both the cumulative ROC and ROC-50 scores, and the differences to the eMOTIF kernel are significant for the cumulative ROC-score ($p = 4 \cdot 10^{-4}$ and $p = 0.7$ on ROC and ROC-50 results with signed rank tests corrected for multiple testing). The results also indicate that extending the eMOTIF kernel with GP motifs improves the performance of the eMOTIF kernel. Even though the performance differences between the GPextended and eMOTIF kernel are not significant ($p = 0.3$ and $p = 0.9$ on ROC and ROC-50), the GP motifs boost the performance of the eMOTIF kernel such that the GPextended

kernel's ROC-scores are not significantly worse than those of the GPkernel ($p = 0.06$).

As Figure 4 shows, the gain of adding additional motifs to the eMOTIF kernel is more evident on the fold benchmark. Because most of the eMOTIFs are relatively specific, the sequences that belong to a fold will on average share few eMOTIFs, giving a very sparse kernel. This might explain the huge performance drop for the eMOTIF method compared with its performance on the superfamily benchmark. If the eMOTIF method has a lack of a suitable set of eMOTIFs for fold detection, the additional motifs made for the GPextended kernel can compensate for this. Both the GPextended kernel's ROC and ROC-50 scores are significantly better than the eMOTIF scores ($p = 2 \cdot 10^{-4}$ and $p = 2 \cdot 10^{-4}$).

The GPkernel has a very good performance on fold detection compared to the other motif methods (all p-values $\leq 10^{-8}$). The key to the GPkernel's increased performance are the motifs trained on the different negative folds. When we tested a kernel that consisted of an equal number of positive motifs only, the average ROC-50 score fell by 30% (data not shown). Similarly, GPboost and the GPextended kernel only use motifs trained to recognize the positive training set and are much less accurate than the GPkernel is.

As the above experiments have shown, the negative motifs are more useful on the fold than on the superfamily recognition problem. Because the positive sequences are more similar on the superfamily than on the fold benchmark, methods that only focus on recognizing the positive sequences can more easily find motifs that characterize the positive sequences than they can on the fold benchmark. On the fold benchmark, the motifs that characterize the positive sequences do not confidently predict a protein's correct fold, but an absence of motifs common to some of the negative folds may complement the occurrence of positive motifs. This complementarity probably explains the GPkernel's higher relative performance on the fold than on the superfamily benchmark compared to the other motif methods.

The GPkernel has better overall performance than existing methods

To further assess the GPkernel's performance, we evaluated the performance of three other popular methods for remote homology detection; PSI-BLAST and the Mismatch and SVM-Pairwise kernels. Figure 5 summarizes the performances of the four methods on the superfamily benchmark. Again, the GPkernel is significantly better than the other methods in terms of ROC scores (p-values of 0.001, 0.0004, and $< 10^{-10}$ for Mismatch, SVM-Pairwise, and PSI-BLAST). The GPkernel also has significantly higher ROC-50 scores than Mismatch and PSI-BLAST ($p = 0.03$ and $p < 10^{-10}$), but the GPkernel and SVM-Pairwise's ROC-50 scores are not significantly different ($p = 0.7$).

Figure 6 shows how the methods compare on the fold benchmark. As would be expected, there is a bigger

difference between the methods when the level of sequence similarity is very low. Especially, the BLAST-based methods have difficulties producing effective alignments between related sequences at the fold level. SVM-Pairwise has a much lower performance on the fold than on the superfamily benchmark, and the scores for PSI-BLAST on fold detection are not reported due to the very poor results achieved. The mismatch kernel, using more general string patterns than the eMOTIF kernel has a stable performance on both benchmarks and is significantly better than SVM-Pairwise on the fold benchmark ($p = 6 \cdot 10^{-8}$ and $p = 1 \cdot 10^{-5}$ for ROC and ROC-50). The GPkernel has the best performance on the fold benchmark; significantly better than the second best performing Mismatch kernel ($p = 2 \cdot 10^{-4}$ and $p = 7 \cdot 10^{-3}$ for ROC and ROC-50). Table 1 gives the average ROC and ROC-50 scores for all of the methods.

Motif based classifiers for fold detection perform better with many motifs of low specificity

One of the SCOP superfamilies (b.68.1) that participate as a test set in the fold detection benchmark is classified well with the GPkernel method (ROC-50 score of 0.903) but achieves a lower score with the eMOTIF method (0.128). Even though there seems to be a mild correlation (0.16) between the number of eMOTIF matches for a fold and the ROC-50 score achieved, the training and test sets for this superfamily do not have significantly fewer eMOTIF matches than other superfamilies. More important is the number of eMOTIFs shared between sequences. This number varies a lot between different pairs of sequences, but if we calculate the average number of eMOTIFs shared between sequences in the b.68 fold, we find that the sequences on average share 0.73 eMOTIFs. This is less than the average for all folds (2.41) which again is much less than the average shared between sequences of a superfamily (11.92). This shows that because sequences at the fold level have a very low sequence similarity, and because of the specificity of most of the eMOTIFs, the number of eMOTIFs shared between sequences in a fold will also be low. This will in turn influence the performance of the eMOTIF kernel.

Table 2 shows examples of GP motifs trained on the training set for the b.68.1 fold classifier. The GP motifs varies and do in general not share any huge similarities with the eMOTIFs that match the sequences of the fold. The table also shows the relative percentage of matches in the positive and negative training and test sets for the fold. The GP motifs do match a higher percentage of the positive sequences than the negatives, but the considerable number of negative sequences that are matched shows the difficulty of finding simple discrete sequence motifs that cover many sequences of a fold while also being as specific as possible. The best GP motifs tend to be either very short sequences or very long complicated expressions with multiple alternative amino acids at each position in the motif.

When looking at all motifs made, we find that each motif on average matches nearly a fourth of all sequences. All sequences will therefore share many GP motifs. If we compare the related sequences of a superfamily or fold with randomly chosen sequences, we find that related sequences share more motifs than randomly chosen sequences. On average, sequences in a superfamily have a higher correlation in their motif matches than other sequences on the positive motifs (correlation coefficient 0.25 versus 0.16) and they also correlate more on the other motifs (0.33 versus 0.21). This means that related sequences share more of all motifs than unrelated sequences, explaining the GPkernel's performance.

Another kernel that also has a good performance on fold detection is the mismatch kernel. This kernel is based on a much larger feature set of even more unspecific patterns than the GPkernel. For the mismatch kernel, the generality of the patterns ensures that the whole solution space of sequences is covered and that most sequences share at least a few patterns. The GPkernel achieves good coverage by training a certain amount of motifs for each superfamily or fold. The GP motifs, while not being too specific, are still more tuned to discriminate between sequences of different folds than the completely general mismatch k-mer patterns. This suggests that to capture the small sequence similarity that exists at the fold level, motif-based classifiers benefit from motifs that are general enough to match a significant number of the weakly similar sequences of a fold. In summary, it seems that good motif-based classifiers on the fold recognition problem need to strike a balance between specificity (eMOTIF) and generality (mismatch). The GPkernel is one step in that direction.

Conclusion

We have introduced a motif kernel with discrete sequence motifs trained with genetic programming. Motifs are evolved using a subset of regular expressions to describe sequences in a superfamily or fold, and discriminate between these and sequences in other superfamilies (folds). The method gives very good results on two SCOP benchmarks when compared to other relevant methods.

In addition, we have established two new and updated benchmark sets. These sets, which are nearly twice as large as previously used benchmarks, should prove useful for future studies on remote homology detection.

Methods

Genetic programming

Genetic programming [25] is a form of automatic programming that aims to find an optimal solution to a problem using a population of candidate solutions and techniques inspired by biological evolution. In genetic programming, the solutions are usually variable sized syntax trees whose structure is defined by the solution language. An example of such a language is regular expressions where the set of terminals are the 20 amino acid characters.

Our algorithm, which is based on the GP-component of the GPboost algorithm [32], uses a standard tree-based representation of individuals. It uses subtree swapping crossover, tree generating mutation and reproduction as genetic operators and uses tournament selection to select individuals for the next generation.

Motif kernels

A motif kernel gives a sequence similarity measure based on the motif content of a pair of sequences [17]. A sequence x can in this context be represented in a vector space indexed by a set of motifs M as $\Theta(x) = (\theta_m(x))_{m \in M}$. In the eMOTIF kernel, $\theta_m(x)$ is the number of occurrences of the motif m in x. The motif kernel is then defined as a linear kernel over the motif contents: $K(x, x') = \Theta(x) \cdot \Theta(x')$. In most cases a motif appears only once in a sequence so this kernel essentially counts the number of motifs that are common to both sequences. This is always the case for the GPkernel, as $\theta_m(x)$ here is 1 if the motif occurs in x and 0 otherwise.

Results benchmarking

To benchmark our method we simulate the process of remote homology detection and fold detection by using the SCOP database [24] as a basis for two benchmarks. The SCOP database aims to classify all proteins of known structure in a hierarchy based on structural and evolutionary relatedness. At the lowest level of the hierarchy, proteins clustered in a SCOP family have clear evolutionary relationship, meaning that pairwise residue identities between proteins are 30% and greater. Proteins in SCOP superfamilies show low degrees of sequence identities, but structural and functional features in the proteins give them a probable common evolutionary origin, meaning that proteins clustered in superfamilies are likely to be homologues. Proteins have the same common fold if they have the same major secondary structures in the same arrangement and with the same topological connections. This does not necessarily mean they have

the same evolutionary origin.

The first benchmark is the now classic benchmark where the goal is to classify a new protein sequence to the correct SCOP superfamily. Here, one family in the superfamily is kept as a positive test set. The other families in the same superfamily constitute the positive training set. The negative test set is made up of one random family from all the other superfamilies and the negative training set has the rest of the families in these superfamilies. Figure 7 illustrates the setup for this benchmark.

For the other benchmark, we follow that of Rangwala and Karypis [16]. We move up one level in the SCOP hierarchy; the objective is to classify an unknown sequence to the correct fold. One superfamily is used as positive test set, while the others in the same fold constitute the positive training set. Negative test and training set are made from the superfamilies of the other folds. This benchmark is considerably harder as most of the sequences within a fold have a very low degree of similarity.

We use sequences from SCOP version 1.67, filtered with Astral [34] to remove sequences that share more than 95% similarity. The data are further filtered according to the principle that each classifier should have at least 10 sequences for testing and training, that is, every classifier should have at least 10 sequences in its positive training and test set. For the superfamily benchmark, this leaves us with 4019 sequences in 392 SCOP families. 102 of these families match the conditions above. The fold benchmark has 3840 sequences from 374 superfamilies and classifiers are made for 86 of these. Of the 3840 sequences in the fold benchmark, 2076 do not participate in the superfamily benchmark. Note that the 102 families and 86 superfamilies tested in our superfamily and fold benchmarks are almost twice the number of families and superfamilies used in previous benchmark studies.

Statistical tests

To determine whether two methods have statistically different ROC or ROC-50 scores on a particular benchmark, we use signed rank tests. All p-values reported are double-sided p-values that have been Bonferroni-corrected for multiple comparisons.

Other methods

We computed the eMOTIF kernel based on the eMOTIFs generated from version 1.0 of the eBLOCKS database. This database contains 522.321 motifs and is the same that was used in the original article [17]. The mismatch kernel is computed by extracting all subsequences of length 5 from the dataset and using the Interagon PMC to search for these subsequences in the data sets, allowing for one mismatch.

We use Clustal version 1.83 to create a multiple alignment of the positive training set and give this as input to PSI-BLAST version 2.2.13. PSI-BLAST is then run with standard parameter values for 1 iteration against the test set. The e-value of the resulting alignments are used to rank the test set. SVM-pairwise is calculated by using the negative logarithm of pairwise BLAST e-values to generate a radial basis kernel with the same parameters as in the original article of Liao and Noble [10].
The Gist package version 2.2 [35] is used to train and test the kernels. Because the test sets have many more negative than positive instances, simply measuring error-rates will not give a good evaluation of performance. Instead we evaluate our results by computing the ROC and ROC-50 scores [36]. A ROC curve is a plot of a classifier's sensitivity as a function of its specificity for different classification thresholds. The ROC score is the area under the ROC curve. The ROC-50 curve is the same as a ROC curve, except the positives are only counted up to the first 50 negatives.

Authors contributions

TH did the experiments and drafted the manuscript. AJHH did the initial work on GPboost for remote homology detection. PS conceived the GPkernel and the study and helped prepare the final version of the manuscript. All authors read and approved the final manuscript.

Acknowledgements

We thank A. Ben-Hur for providing his eMOTIF kernel code and O. Snøve Jr. for useful comments on the manuscript. PS receives support from the National Program for Functional Genomics in Norway (FUGE) and the Leiv Eriksson program of the Norwegian Research Council.

References

1. Smith T, Waterman MS: **Identification of common molecular subsequences**. *J Mol Biol* 1981, **147**:195–197.
2. Altschul SF, Madden TL, Schaffer AA, Zhang J, Zhang Z, Miller W, Lipman DJ: **Gapped BLAST and PSI-BLAST: A new generation of protein database search programs**. *Nucleic Acids Res* 1997, **25**:3389–3402.
3. Pearson WR: **Rapid and sensitive sequence comparisons with FASTP and FASTA**. *Methods Enzymol* 1990, **183**:63–98.
4. Gribskov M, Lüthy R, Eisenberg D: **Profile analysis**. *Methods Enzymol* 1990, **183**:146–159.
5. Krogh A, Brown M, Mian I, Sjolander K, Haussler D: **Hidden Markov models in computational biology: Applications to protein modelling**. *J Mol Biol* 1994, **235**:1501–1531.
6. Baldi P, Chauvin Y, Hunkapillar T, McClure M: **Hidden Markov models of biological primary sequence information**. *Proc Natl Acad Sci U S A* 1994, **91**:1059–1063.

7. Karplus K, Barrett C, Hughey R: **Hidden Markov models for detecting remote protein homologies**. *Bioinformatics* 1998, **14**(10):846–856.

8. Corinna C, Vapnik V: **Support-Vector Networks**. *Mach Learn* 1995, **20**(3):273–297.

9. Jaakkola T, Diekhans M, Haussler D: **Using the Fisher kernel method to detect remote protein homologies**. In *Proc Int Conf Intell Syst Mol Biol.*. Edited by Lengauer T, Schneider R, Bork P, Brutlag DL, Glasgow JI, Mewes HW, Zimmer R, AAAI 1999:149–158.

10. Liao L, Noble WS: **Combining pairwise sequence similarity and support vector machines for detecting remote protein evolutionary and structural relationships**. *J Comput Biol* 2003, **6**(10):857–868.

11. Hou Y, Hsu W, Lee LM, Bystroff C: **Efficient remote homology detection using local structure**. *Bioinformatics* 2003, **17**(19):2294–2301.

12. Hou Y, Hsu W, Lee LM, Bystroff C: **Remote homolog detection using local sequence-structure correlations**. *Proteins* 2004, **57**:518–530.

13. Leslie CS, Eskin E, Cohen A, Weston J, Noble WS: **Mismatch string kernels for discriminative protein classification**. *Bioinformatics* 2004, **20**:467–476.

14. Kuang R, Ie E, Wang K, Wang K, Siddiqi M, Freund Y, Leslie C: **Profile-Based String Kernels for Remote Homology Detection and Motif Extraction**. *Proc IEEE Comput Syst Bioinform Conf* 2004, **00**:152–160.

15. Saigo H, Vert JP, Ueda N, Akutsu T: **Protein homology detection using string alignment kernels**. *Bioinformatics* 2004, **20**:1682–1689.

16. Rangwala H, Karypis G: **Profile-based direct kernels for remote homology detection and fold recognition**. *Bioinformatics* 2005, **21**(23):4239–4247.

17. Ben-Hur A, Brutlag D: **Remote homology detection: a motif based approach**. *Bioinformatics* 2003, **19**:i26–33.

18. Sigrist C, Cerutti L, Hulo N, Gattiker A, Falquet L, Pagni M, Bairoch A, Bucher P: **PROSITE: a documented database using patterns and profiles as motif descriptors**. *Brief Bioinform* 2002, **3**:265–274.

19. Attwood T, Bradley P, Flower D, Gaulton A, Maudling N, Mitchell A, Moulton G, Nordle A, Paine K, Taylor P, Uddin A, Zygouri C: **PRINTS and its automatic supplement, prePRINTS**. *Nucleic Acids Res* 2003, **31**:400–402.

20. Henikoff S, Henikoff J, Pietrokovski S: **Blocks+: a non-redundant database of protein alignment blocks derived from multiple compilations**. *Bioinformatics* 1999, **15**(6):471–479.

21. Mulder NJ, Apweiler R, Attwood TK, Bairoch A, Bateman A, Binns D, Bradley P, Bork P, Bucher P, Cerutti L, Copley R, Courcelle E, Das U, Durbin R, Fleischmann W, Gough J, Haft D, Harte N, Hulo N, Kahn D, Kanapin A, Krestyaninova M, Lonsdale D, Lopez R, Letunic I, Madera M, Maslen J, McDowall J, Mitchell A, Nikolskaya AN, Orchard S, Pagni M, Ponting CP, Quevillon E, Selengut J, Sigrist CJA, Silventoinen V, Studholme DJ, Vaughan R, Wu CH: **InterPro, progress and status in 2005**. *Nucleic Acids Res* 2005, **33**(Database Issue):D201–205.

22. Nevill-Manning CG, Wu TD, Brutlag DL: **Highly Specific Protein Sequence Motifs for Genome Analysis**. *Proc Natl Acad Sci U S A* 1998, **95**(11):5865–5871.

23. Su QJ, Lu L, Saxonov S, Brutlag DL: **eBLOCKS: enumerating conserved protein blocks to achieve maximal sensitivity and specificity**. *Nucleic Acids Res* 2005, **33**:D178–D182.

24. Murzin AG, Brenner SE, Hubbard T, Chothia C: **SCOP: a structural classification of proteins database for the investigation of sequences and structures.** *J Mol Biol* 1995, **247**:536–540.

25. Koza J: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press 1992.

26. Koza J, Andre D: **Automatic Discovery of Protein Motifs Using Genetic Programming**. In *Evolutionary Computation: Theory and Applications*. Edited by Yao X, World Scientific 1996.

27. Yuh-Jyh H: **Biopattern discovery by genetic programming**. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*. Edited by Koza JR, Banzhaf W, Chellapilla K, Deb K, Dorigo M, Fogel DB, Garzon MH, Goldberg DE, Iba H, Riolo R, University of Wisconsin, Madison, Wisconsin, USA: Morgan Kaufmann 1998:152–157.

28. Seehus R, Tveit A, Edsberg O: **Discovering Biological Motifs With Genetic Programming**. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, Volume 1. Edited by Beyer HG, O'Reilly UM, Arnold DV, Banzhaf W, Blum C, Bonabeau EW, Cantu-Paz E, Dasgupta D, Deb K, Foster JA, de Jong ED, Lipson H, Llora X, Mancoridis S, Pelikan M, Raidl GR, Soule T, Tyrrell AM, Watson JP, Zitzler E, Washington DC, USA: ACM Press 2005:401–408.

29. Olsson B: **Using evolutionary algorithms in the design of protein fingerprints**. In *Proceedings of the Genetic and Evolutionary Computation Conference*, Volume 2. Edited by Banzhaf W, Daida J, Eiben AE, Garzon MH, Honavar V, Jakiela M, Smith RE, Orlando, Florida, USA: Morgan Kaufmann 1999:1636–1642.

30. Ross BJ: **The Evolution of Stochastic Regular Motifs for Protein Sequences**. *New Generation Comput* 2002, **20**(2):187–213.

31. Halaas A, Svingen B, Nedland M, Sætrom P, Snøve Jr O, Birkeland OR: **A Recursive MISD Architecture for Pattern Matching**. *IEEE Trans on VLSI Syst* 2004, **12**(7):727–734.

32. Sætrom P: **Predicting the efficacy of short oligonucleotides in antisense and RNAi experiments with boosted genetic programming**. *Bioinformatics* 2004, **20**(17):3055–3063.

33. Meir R, Rätsch G: **An introduction to boosting and leveraging**. In *Advanced Lectures on Machine Learning*, Volume 2600. Edited by Mendelson S, Smola A, Springer-Verlag 2003:118–183.

34. Brenner S, Koehl P, Levitt M: **The ASTRAL compendium for sequence and structure analysis**. *Nucleic Acids Res* 2000, **28**:254–256.

35. Pavlidis P, Wapinski I, Noble WS: **Support vector machine classification on the web**. *Bioinformatics* 2004, **20**:586–587.

36. Gribskov M, Robinson NL: **Use of receiver operating characteristic (ROC) analysis to evaluate sequence matching**. *Comput Chem* 1996, **20**:25–33.

Figures

Figure 1 - Splitting of training sets for GP algorithm

The figure shows three families from an example superfamily that constitute the positive training set for GP. The positive test set and the negative training and test set are not shown in this figure. In addition to train motifs to cover the sequences of all three families (large broken blue oval), we also train GP on all the possible subsets of this superfamily that exclude one family of the positive training set. This is indicated by the red, green, and yellow dashed ovals in the figure. Ten motifs are made for each subset.

Figure 2 - Boosted classifiers

The figure shows the setup of the boosted classifiers. For each test set, we create 10 boosted classifiers whose predictions are averaged to give a final classification. Each classifier is made from 100 boosted GP motifs.

Figure 3 - GPkernel has highest overall performance of motif methods on superfamily benchmark.

The graphs show the cumulative number of families with a ROC (top) and ROC-50 (bottom) score greater than a given value for the GPkernel, GPboost, GPextended, and eMOTIF methods.

Figure 4 - GPkernel outperforms other motif methods on SCOP fold benchmark

The graphs plot the cumulative number of superfamilies with a ROC (top) and ROC-50 (bottom) score greater than a given value for the GPkernel, GPboost, GPextended and eMOTIF methods.

Figure 5 - GPkernel compares favorably to common methods for remote homology detection on superfamily benchmark.

The figure plots the cumulative number of families with a ROC (top) and ROC-50 (bottom) score greater than a given value for the GPkernel, eMOTIF, Mismatch, SVM-Pairwise and PSI-BLAST methods.

Figure 6 - Large differences in performance on fold detection

Figure 6 plots the cumulative number of superfamilies with a ROC (top) and ROC-50 (bottom) score greater than a given value for the GPkernel, eMOTIF, Mismatch and SVM-Pairwise methods. PSI-BLAST is omitted in this context due to the method's very poor results on fold detection.

Figure 7 - SCOP superfamily benchmark

The figure shows how the SCOP database is divided into training and test sets. For each classifier tested on the superfamily benchmark, the sequences of the SCOP database are divided into positive and negative training and test sets. One SCOP family is used as a positive test set. The negative test set is made from one random family from each of the other superfamilies. The positive training set is made from the superfamily of the classifier, excluding the positive test set family. The negative training set is made from the other superfamilies, excluding the negative test sets.

Tables

Table 1 - Average ROC/ROC-50 scores

Table 1 shows the average ROC and ROC-50 scores obtained by the different methods on the superfamily benchmark and the fold benchmark.

Average ROC and ROC-50 scores				
	Superfamily level		Fold level	
	ROC	ROC-50	ROC	ROC-50
GPkernel	0.902	0.590	0.844	0.514
GPextended	0.869	0.542	0.753	0.371
GPboost	0.797	0.375	0.688	0.298
SVM-Pairwise	0.849	0.555	0.724	0.359
Mismatch	0.878	0.543	0.814	0.467
eMOTIF	0.857	0.551	0.698	0.308
PSI-BLAST	0.575	0.175	0.501	0.010

Table 2 - Examples of GP motifs

The table shows examples of the motifs evolved by the genetic programming process targeting the SCOP b.68 fold. In addition to amino acid characters, the motifs are also made from the disjunction operator (|), the wildcard operator (.), and the Hamming distance operator $\{: p >= x\}$ that specifies the minimum number of characters that must match in the pattern. For each motif, the table shows the relative percentage of sequences matched in the training and test sets. The positive training set has 12 sequences, the negative 3590 sequences. The positive test set also has 12 sequences; the negative set has 226 sequences.

Examples of GP motifs				
Motif	PTr	NTr	PTe	NTe
$\{MEEIEII : p >= 3\}$	67	41	67	55
$\{IQIIIEE : p >= 3\}$	83	38	92	50
$\{(I\|I)E(E\|(I\|E)) : p >= 4\}$	58	37	83	51
$\{TQ(D\|H)(K\|C)(D\|H)((((D\|H)\|A)\|A)\|A)TQ((H\|A)\|A)TQ((D\|H)\|A)(I\|A) : p >= 7\}$	33	20	33	23
$\{M(L\|L)CARACAARAA(L\|L)RACAA : p >= 6\}$	8	28	50	44
$\{AALAALA(A\|M)AA.ILAL(A\|M)AA(C\|M)AV.IL(.\|T)A.ILAAALA(.\|(A\|M))V.ILVAA.ILL(.\|T).IA(A\|M)AALA(A\|M)V.ILV(R\|M) : p >= 20\}$	50	28	25	45
$\{(L\|(M\|A))(L\|(L\|(M\|A)))(L\|(M\|A))(L\|((L\|A)\|A))M : p >= 5\}$	83	37	67	54

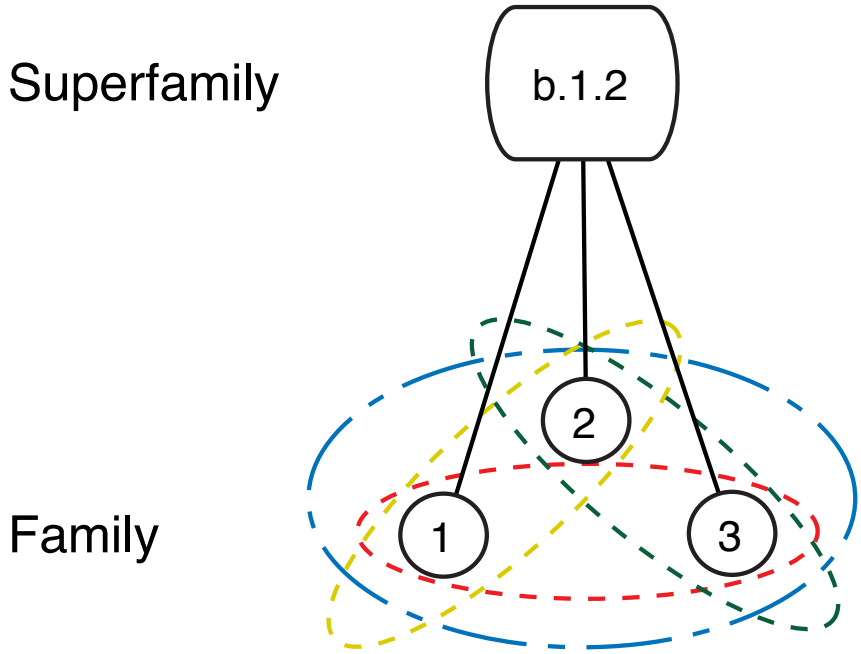

Figure 1: Figure 1

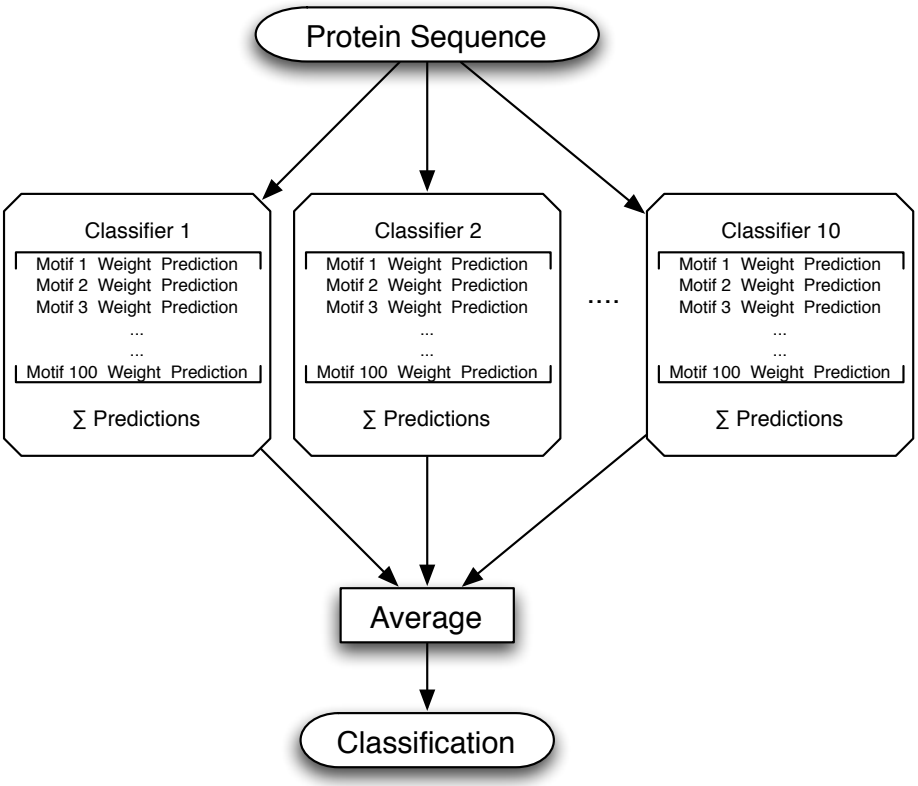

Figure 2: Figure 2

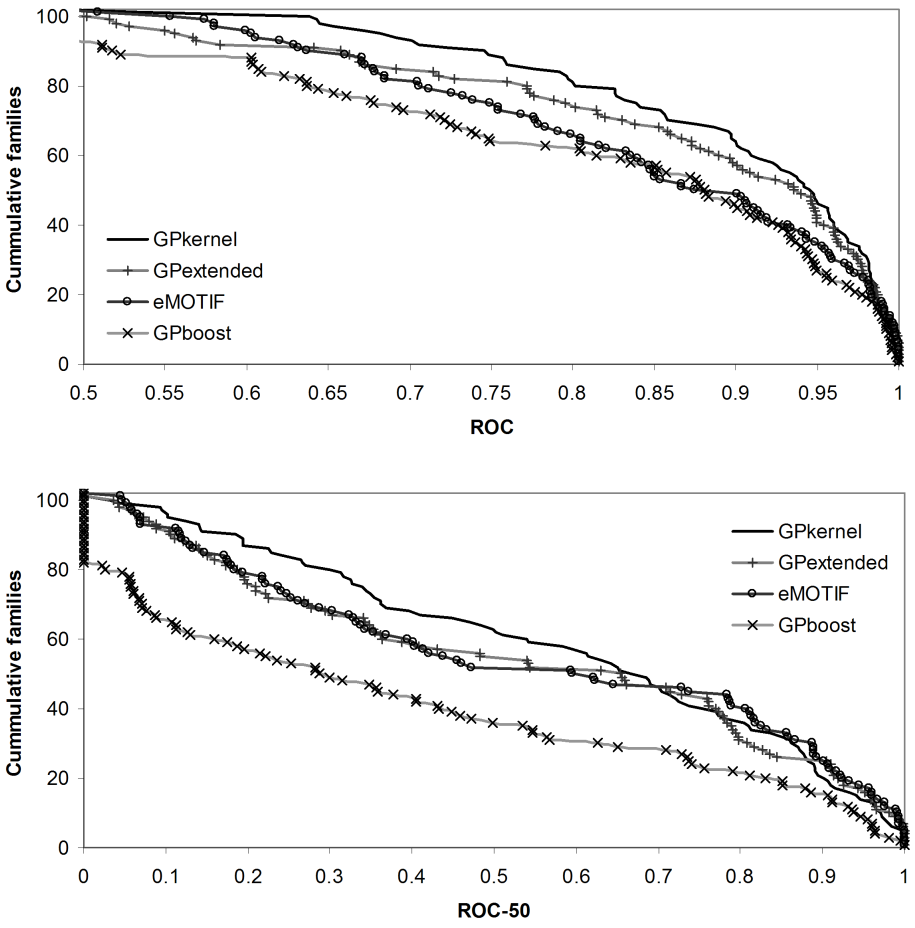

Figure 3: Figure 3

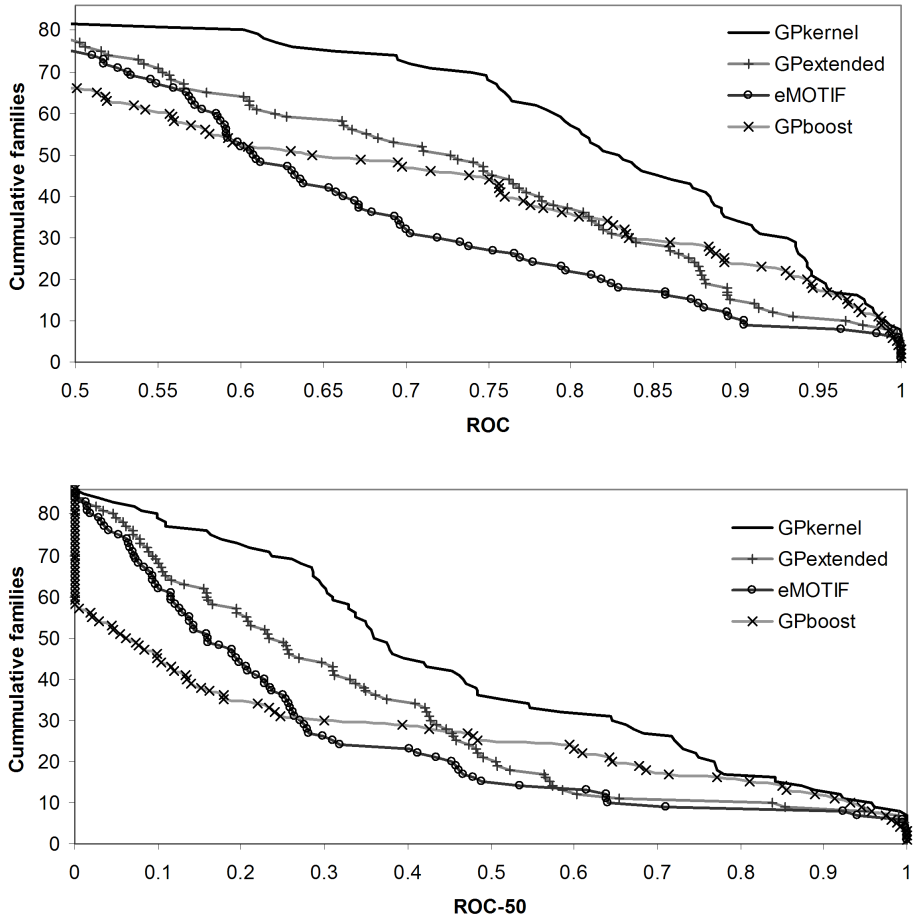

Figure 4: Figure 4

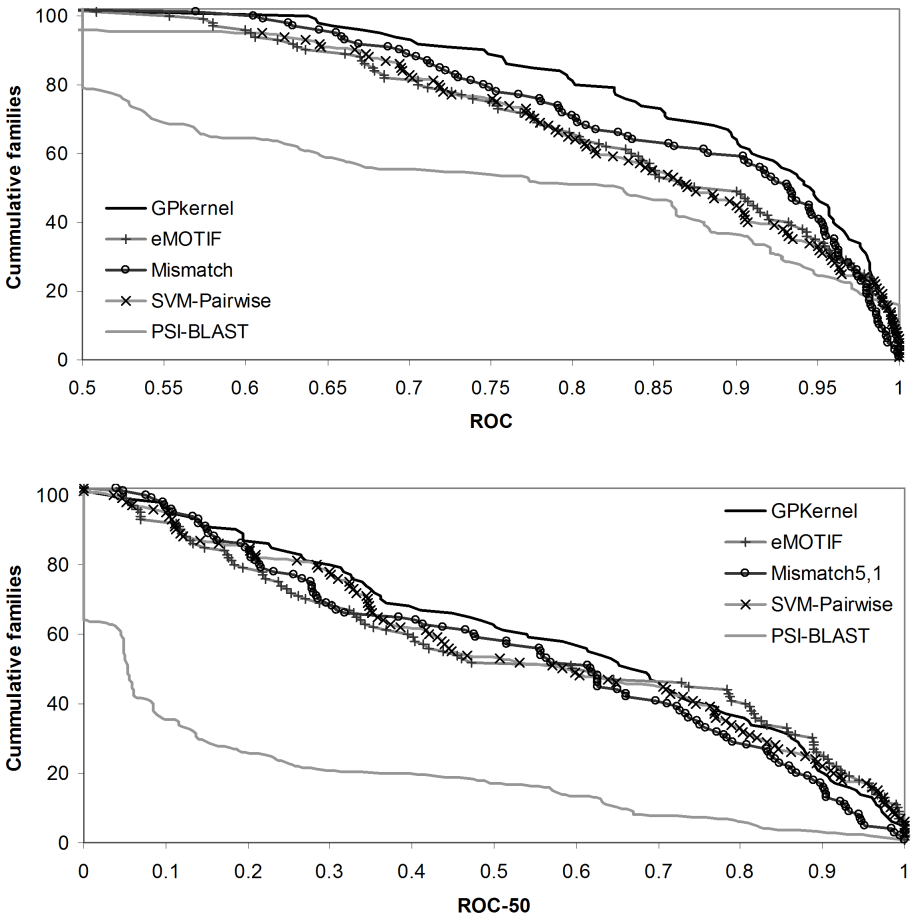

Figure 5: Figure 5

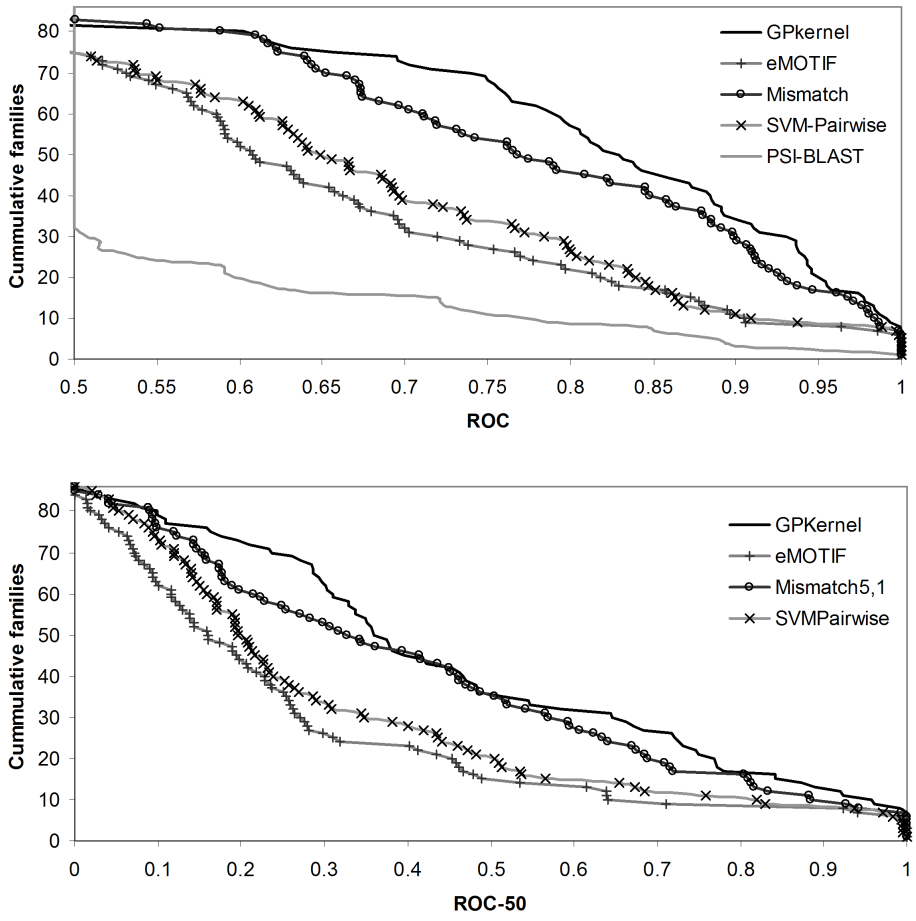

Figure 6: Figure 6

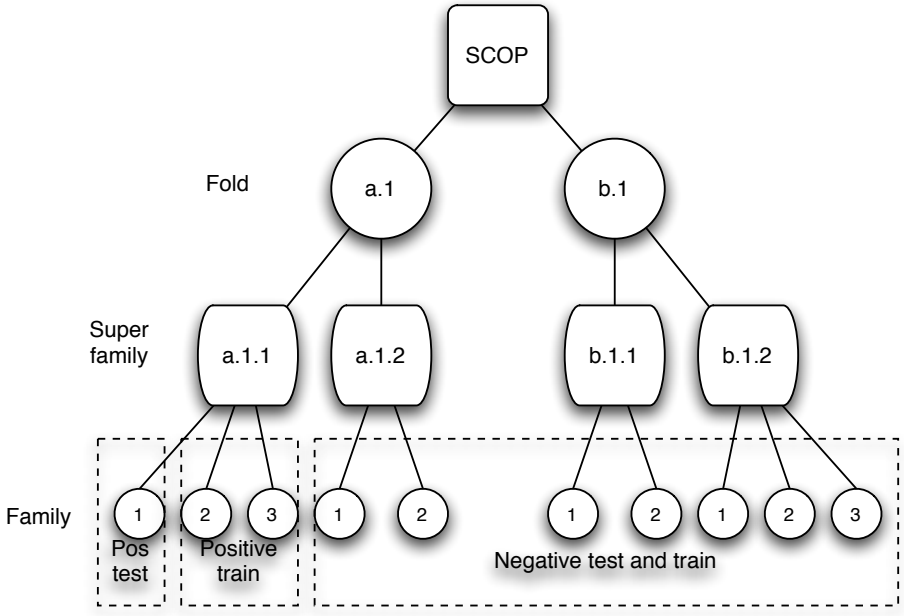

Figure 7: Figure 7