Radboud University Nijmegen

Master thesis in information science

Relief-based feature selection in bioinformatics: detecting functional specificity residues from multiple sequence alignments

Author:
Wout Megchelenbrink

First supervisor:
Dr. Elena Marchiori

Second supervisor:
Dr. Peter Lucas

Thesis number:
132IK

Abstract

Finding functionality specific residues in proteins is important for a better understanding of many biological processes. A better knowledge of these amino-acids can help understanding the development of certain diseases and aid in drug design. Finding these residues is not easy and biological experiments take a lot of time, resulting in large costs. Automated computing of these residues from multiple sequence alignments (MSA) can aid biologists in this task. It is generally assumed that conservation of residues within a subfamily of a MSA and divergence of this residue between subfamilies is a strong lead for functional specificity. Many feature selection algorithms have been developed to exploit this difference. Of these algorithms, Relief-based methods generally show good performance because they perform a global and local search based on a nearest neighbor classifier.

Relief-based algorithms compute a feature weight W_i for every residue i. Consider the simple case where a MSA is divided into two subfamilies. Relief computes the weight by finding the nearest neighbor of a sample X from the same subfamily (the nearest hit, NH) and one from the opposite subfamily (the nearest miss, NM). In this case, because I use MSA's, the Hamming distance is used as a distance metric. Now the weight can be computed as:

$$W_i = W_i + \alpha_0|X_i - NM_i| - \alpha_1|X_i - NH_i|, \tag{1}$$

where $|\cdot|$ is the distance between residues. Weights are computed by iterating over all samples. Relief uses $\alpha_0 = \alpha_1 = 1$, which means that within-class conservation and inter-class divergence are 'weighted' equally. In this thesis I tried to optimize the predictive performance of several Relief-based algorithms by increasing the value of α_1 thereby emphasizing conservation. All algorithms used can handle multi-class data. Experiments have been conducted on 18 datasets. The residues predicted as specificity determining have been compared with 'known positives' from biological experiments using the area under the ROC-curve. Results show a trend where datasets containing proteins of short length (< 300 amino-acids) benefit from this distinction, whereas datasets containing large proteins (> 1000 amino-acids) show no improvement.

Preface

This master thesis about feature selection in bioinformatics consists of eight chapters. The first chapter is a general introduction on the topic of feature selection. Chapter two deals with the characteristics of biological sequence data. In the third chapter, I'll explain the basic theory of feature selection and some approaches to keep in mind when creating new methods. The fourth chapter is about related work on feature selection in bioinformatics. Chapter 5 explains the basics of Relief as a two-class feature selection method. It also discusses some successful algorithms that extend this subject to multi-class data and deal with some of the issues discussed in chapter 2. In chapter 6 I'll discuss a new approach, based on Relief. In this chapter the new theory is discussed and the research method and results are provided. Chapter 7 discusses the results of the experiments conducted. Finally, chapter 8 concludes the results and gives some guidelines for future research.

With this thesis comes a webserver implementation of the algorithms discussed in chapter 5 and 6. This code is available on CD-ROM. Information about the implementation of the webserver can be found in appendix A.

Acknowledgments

I'm very glad that I was given the opportunity to combine my interest and knowledge for computers and programming with my interest for biology. I like new challenges and bioinformatics certainly is a challenging and interesting research terrain. I'd like to specially thank my supervisor Elena Marchiori for giving me the opportunity to study this topic, who helped me with questions and provided useful feedback. I'd also like to thank my girlfriend Jolien for helping me with the ungrateful task of putting all data in a spreadsheet to obtain the tables and graphs in this thesis.

Nijmegen, July 2010

Contents

List of Tables

List of Figures

Chapter 1

Introduction to feature selection

1.1 Introduction

Fast improvements in research and ongoing developments in information technology enables us to collect and store huge amounts of data. The Internet with all its webpages is probably the most well-known example of the explosion of information. To find your information, many search engines exist to aid you in this task. Biology is no exception to the domains where technology contributed to a tremendous increase in available data. You can think of the sequencing of DNA and proteins, yielding in databases with thousands to billions of records. Understanding these building blocks of life is very important, for example for the production of food, fighting and preventing diseases, drug discovery, understanding the genome and better understanding evolution. But finding out which genes are involved for instance in the development of diseases is not an easy task. There are millions of nucleotides in the human genome and in most cases, there are multiple factors and genes involved in the development of these diseases.

Any disease affecting or affected by the genome can potentially be discovered. After that, effective drugs or treatment to solve or prevent this disease have to be discovered. In the case of drug discovery, analysis of all the residues in a protein family is virtually impossible and automated feature selection can help out.

The problem is not unique to biology as well, for instance the Internet has billions of webpages. Billions of e-mail messages are being send each day, of which a large quantity is spam. Newsgroups, forums, auctioning websites and social networking sites such as Facebook, YouTube and Twitter emerge quickly. In many of these cases, people want to search and categorize huge amounts of data. Data with many features, of which only a small part is relevant. The same goes for research data and datamining. In all

fields where people want to automatically deduce relevant information from high dimensional data, feature selection can offer a helping hand. So it is not strange that computing science can use the lessons learned from these domains and use them for other research areas, such as bioinformatics.

In this thesis, I will focus on the biological datasets of multiple sequence alignments (MSA). By using Relief-based algorithms, I will try to select functional specificity determining residues in the MSA's (see chapter 2 for more information about sequence data and families) of families of these large molecules.

1.2 What is feature selection?

Feature selection is about selecting relevant features from a high dimensional space. This sounds somewhat abstract and without much knowledge of biology, it doesn't get any easier. To get a basic understanding of the principles of feature selection, let's look at a very basic example. How this is applied to biological data is discussed in the next chapter.

Example 1.2.1 Feature selection on colored cards
Imagine you have four colored cards, with a three letter code written on it. The colors and codes are given in table 1.1 and are independent of each other.

Card Nr	Color	Code
1	Blue	AB
2	Blue	BA
3	Red	AC
4	Red	CA

Table 1.1: Simple feature selection example

Suppose we have two boxes: R and B. When I ask you to put the blue cards in box B and the red ones in the box R, you will implicitly select the relevant features for these cards, namely the color, and discard the code on the card.
Now we make it somewhat more difficult. I say there is a price on the backside of the card. The price can be either 0 or 100 euro. I don't tell you which ones it is, but I give you table 1.2:

To solve this problem, you will probably do the following. First you'll -unconsciously- make two classes; one without a price ($class_{np}$) and the other with a price ($class_p$). Then you look for a common pattern in $class_p$. You find out that there is a price for both blue cards and red cards, so color is no good. You look at the code. You notice all cards with a price contain

Card Nr	Color	Code	Price
1	Blue	AC	100
2	Blue	BA	0
3	Blue	CB	0
4	Blue	BC	100
5	Red	AB	0
6	Red	BA	0
7	Red	BC	100
8	Red	CB	0

Table 1.2: Feature selection with a price

letter C. But you see that letter C can also be found in $class_{np}$*. You look at the position of the C and find that all prices have C in second position, and if C is in second position, you observe there is always a price. So a C in second position is a good feature; color is not and the complete code also doesn't tell you much. In other words, you'll only have to look for a letter C in second position of the code and find that card number 3 is probably the winning card in table 1.1.*

1.3 Feature selection in bioinformatics

Bioinformatics is the application of information technology and computer science to the field of molecular biology [1]. Bioinformatics is about using computer science, machine learning, pattern recognition and the like to discover the mechanisms in molecular biology. Bioinformatics covers many areas, some profound examples are sequence alignments, splice-site prediction and discovering gene expression using microarrays. Feature selection is important in virtually all areas of bioinformatics, because the enormous amount of data doesn't allow inferring information easily. You'll often have to deal with high dimensional data (genomic data with thousands to ten-thousands of nucleotides) and small sample sizes [27]. Bringing this dimensions down, and selecting only the relevant aspects, is what feature selection is all about.

In this thesis I will focus on feature selection in biological sequences; MSA's of some well-studied protein families to be exact. More information about what bioinformatics is and is not can be found on Wikipedia and a simplified but nevertheless entertaining paper of Dr. Achuthsankar S. Nair [1].

[1] http://en.wikipedia.org/wiki/Bioinformatics

Chapter 2

Biological sequence data and functionality specific residues

To understand the subject of feature selection in bioinformatics, it is good to have an understanding of what biological data is and some side-effects to bear in mind. This chapter will not deeply elaborate the biological processes, because I am not a biologist. The purpose of this chapter is to provide a basic understanding of the data dealt with throughout this thesis. The last part discusses some challenges to keep in mind while dealing with this kind of data.

2.1 Sequence data

A protein is a large sequence of amino-acids (a amino-acid is also called a residue). In real life, these amino-acids are not nicely packed one after another, but create a difficult folded structure that gives the protein its basic function. Many proteins can bind other molecules and sort of 'capture' them in this folded structure and release it somewhere else. Based on this 'primary' function, proteins can be put in some hierarchical families, such as the G-Protein Coupled Receptors (GPCR) or SMAD family. These families can be divided in subfamilies with a given 'subfunctionality'.

The construction of a protein is a difficult biological process, where polarity, hydrophobicity, charge and other chemical principles play an important role. What is important for specificity computing is that a protein can be seen as a string of amino-acids. In nature there are 20 different amino-acids, which names can be abbreviated with letters from A to W (not all letters are used). Proteins within a family can all be represented with such a string representation. The most well-known string format for MSA's is the FASTA-format, which is just this encoding of an amino-acid to an alphabet letter.

2.1.1 MSA: Multiple Sequence Alignment

Since the late 1980's, there has been an exponential growth in sequence data. This is mainly caused by efficient experimental techniques, like DNA sequencing. To interpret all this data, sophisticated techniques are required. A fundamental feature of chain molecules, like DNA and proteins is that they can be represented in the form of digital symbols, like the letters of our alphabet [3].

Remember that a protein is a difficult folded structure with no clear beginning or end. Thus a string representation has no meaningful beginning or end. If we want to compare proteins in a given family, we want their amino-acids to be as much alike as possible. This is done by aligning the sequences in a family, and the result is known as a multiple sequence alignment (MSA). This process can be seen as shifting the proteins left or right, such that as many amino-acids are the same as possible. Equal amino-acids are named 'conserved' amino-acids or conserved residues. Inserting 'gaps' (sort of padding of no value) ensures that all sequences in the MSA are of equal length. A simplified example from [1] is depicted in figure 2.1

```
G A T E   L I K E S   C H E E S E
|
G R A T E D   C H E E S E
```

(a) Before alignment

```
G - A T E   L I K E S   C H E E S E
|   | | |             |   | | | | |
G R A T E D   - - - - -   C H E E S E
```

(b) After alignment

Table 2.1: Sequence alignment example on English text

Aligning a large amount of sequences is a computational difficult task. Basic local alignment search tool (BLAST) [2] is probably the most used and known tool for this task. Once the sequences are aligned, we can start computing. A more realistic example is figure 2.1. This is a small snippet of a MSA from the GPCR database. Some of the amino acids have a different color, so that differences can be spotted easily.

```
XP_001503773   TERMLISMTLVIITSLTMLLNSAVIMAICTTKKLHQPANYI
NP_001108338   HDRALLVSFLLLFSLTTVFGNMLVILAVVRERYLHTSTNYI
XP_001926072   GSRVILYLVLGFGSLLAVFGNVLVMTSVLHFKQLHSPANFI
XP_001842713   WQTILIAICLAICIILTIGGNILVLLAFIVDRSIRQPSNYI
XP_001666590   VQTVILASVLLLLILSCFIGNLFVILAIIMERDLRGPQYYI
XP_969037      GSRNYWALVLVLFPIFTLFGNVLVILSVYRERTLQSATNYI
NP_001024521   LFQILKGSALFLLVLWTIFANSLVFIVLYKNPRLQTVPNLI
XP_588413      ASRVILYMVYGFGAVLAVFGNLLVMTAILHFKQLHSPTNFI
```

Figure 2.1: A snippet of a MSA from the GPCR database obtained from *http://www.gpcr.org/7tm/*.

2.1.2 Specificity determining residues

In an MSA, we can look at the amino-acids and try to find a pattern for residue position. In finding these specificity determining residues a notion of conservation and divergence is important. The conserved residues are important because they are believed to have a common ancestor in evolution, whereas divergence is a signal of a functional specialization of a residue. *'... the common assumption of conservation of functional residues during evolution'* [23]. This biological phenomenon enables us to make computations, based on this difference.

Residue positions can be fully conserved (the amino-acids at position i in subfamily S of the MSA are all the same) or fully divergent (they are all different). Table 2.2 shows this principle on a toy example. Protein 1-3 are of subfamily A, proteins 4-6 are of subfamily B. Note that position 1 is fully conserved in both subfamilies and is not of much interest. Position 2 is fully conserved in subfamily A and fully divergent between A and B, which indicates that this is an interesting residue. Position 3 is only partly conserved and finally position 4 is conserved *between* the subfamilies. This is not a good indication of functional specificity and so we don't want these kind of residues. In the weighting algorithms, residue 1 is good and will get high weight, residue 4 gives no information and so we penalize this with low (or negative) weights.

	X	1	2	3	4
Subfam. A	1	A	B	C	D
	2	A	B	C	E
	3	A	B	B	F
Subfam. B	4	A	C	B	D
	5	A	C	C	E
	6	A	C	B	F

Table 2.2: Conservation and divergence example: The table shows two subfamilies, each containing 3 proteins with 4 residue positions.

2.2 Challenges in feature selection

Feature selection in bioinformatics is not straightforward. You often have a lot of data with many possible features, but few training examples. I'll list some of the difficulties you often have to cope with in feature selection. Fortunately, there are a lot of papers that try to solve or mitigate these issues. I'll briefly discuss some of these issues now.

- The 'curse of dimensionality' and data sparseness
- Unlabeled data

- Noise or gaps
- Dependent features
- Bias and overfitting

The curse of dimensionality

Example 1.2.1 is of course highly simplified. For instance in bioinformatics, the number of features is much much higher, there are more classes and more instances. On top of that there is often little training data. This means there are lots of possible relevant feature sets, and only little samples to learn the relevant features. This problem is known as the 'curse of dimensionality', introduced by Bellman and illustrated in [21]. It says that a fixed data sample becomes exponentially sparse as the number of dimensions increase, according to the formula:

$$SD \propto M^{1/N}, \tag{2.1}$$

With sampling density SD, M samples and N dimensions. The intuition behind this simple formula is that for every added feature, the possible outcomes increase exponentially. Therefore you also need the number of samples to increase with this rate, to deduce the information.

In real world problems there are not only many more features than in the simple example above. Sometimes you don't have a *good* versus a *bad* class. You have many classes, or worse you don't have any classes at all. Often you have a sparse training set, which doesn't allow you to make a decision with certainty. You have errors or gaps in your training data or features are not independent. So there is a need for algorithms that can cope with these problems and select relevant features, that allow for predicting with high accuracy.

Unlabeled data

Most of the feature selection done today is based on supervised learning. This means the data is labeled because proteins belong to some subfamily. Sometimes you don't have labeled data, because the cost of labeling is too big, it takes too much time, or people simply don't know which subfamily a protein or DNA-string belongs to. Unlabeled data problems occur frequently also, and there is a lot of research for classifiers and feature selection algorithms that can handle unlabeled data. Although this research area, called unsupervised or semi-supervised learning is evolving quickly, in this thesis I will only consider supervised learning.

Noise and gaps

The sequencing and alignment of biological data is not perfect. It's possible that some amino-acids are replaced or the alignment algorithm used

is suboptimal. This can lead to noise (unwanted artifacts) in your MSA's. The very task of aligning sequences leads almost irrevocably to gaps. So algorithms in bioinformatics have to cope with noise and gaps.

Dependent features

Words in a sentence are often not independent. For instance finding the word 'Barack' increases the likelihood of 'Obama'. Thus these words are not independent. However, often treating all words as if they were independent (the so called *bag-of-words* model) yields pretty good classifiers. For speech recognition, this is often not good enough, because you have to make ad hoc predictions. In this case for example consecutive words are often treated as dependent (so called *n-grams* of n consecutive words). The same principle holds for bioinformatics. It's unlikely that amino-acids in a sequence are all independent. However, many models treat them this way. Sequences in biology are not just as linear as they appear in their FASTA-format. DNA for example, is known as a twisted string, or helix. The structure of proteins is even more complicated, with the chemical properties mentioned above. Thus deciding what is a relevant feature in the case of proteomics is not easy. Comparing single amino-acid positions is in fact the naive counterpart of the bag-of-words model, but these algorithms also show good results.

Bias and overfitting

Training data in bioinformatics is often sparse due to high costs of sequencing or because families just don't contain that many proteins. Therefore you'll have to train and test your model with small datasets. Using a method like LOOCV is often inevitable to test the outcome, but results in an increase in computational complexity, because you have to run the algorithm for every sample.

2.3 How can feature selection help?

Feature selection helps reducing input dimensionality. For instance for breast cancer detection, you look at the expression of genes in healthy persons and in breast cancer patients. You try to find significant gene expression differences between these two groups. Of the thousands of genes that could possibly be relevant, good algorithms can bring this back to just a couple, or in the order of tens of genes. This makes it easier for doctors to assess the possibility that someone has breast cancer. There is a lot of development in this area and feature selection algorithms and classifiers are not perfect. However methods improve at fast rate. It's likely that they will assist doctors in diagnosing hereditary diseases at a larger scale in the nearby future.

This breast cancer example is one of the many cases where feature selection shows its importance for real life problems. There are many fields in bioinformatics where this can be applied [27]. It's also applied in completely different fields like spam filtering and text categorization. This shows that feature selection is not only important in science, but can have very important social consequences.

It is clear that feature selection is used for many applications. There are many different algorithms, some slow but accurate, some fast but less accurate. There is wide variety in methods, data used and for instance time complexity. An accurate method for feature selection on small protein datasets might not be desired for large DNA datasets. In general, the choice of algorithm highly depends on the type of problem your dealing with, what results you want (and what not) and the data your dealing with. In general there is no such thing as a 'best' algorithm, but some guidelines are provided by [11].

Chapter 3

Feature selection and learning algorithms

In this chapter I'll describe some of the fundamentals of feature selection. This chapter is by far not exhaustive, but gives a basis for the research and decisions made later on. Liu and Motoda [20] [21] have written some excellent books on feature selection and explained a lot of theory and many common algorithms. Another useful introduction can be found in [10].

3.1 Supervised and unsupervised learning

There are two main types of learning problems: *supervised* and *unsupervised* learning. Supervised learning problems deal with labeled data, in the sense that they take class information into account. Unsupervised learning lacks class information, therefore finding these group labels (known as clustering) is often the purpose of these algorithms. The MSA datasets I use are all labeled, making supervised learning possible.

For the sake of completeness, I'll have to mention that there is also semi-supervised learning where only a few datapoints are labeled. These labeled points can be used to guide the unsupervised learning algorithm. Unsupervised learning is often more difficult but also very interesting.

3.2 Feature relevancy

Feature relevancy is discussed in [13] by Kohavi, John and Pfleger. Blum and Langley [4] have extended some of these thoughts and theorems. Kohavi, John and Pfeger discuss the three basic levels of relevancy:

- Strong relevance
- Weak relevance
- Irrelevance

Strong Relevance
A feature X_i is relevant iff there exists some x_i, y and s_i for which $P(X_i = x_i, S_i = s_i) > 0$ such that $P(Y = y|X_i = x_i, S_i = s_i) \neq P(Y = y, S_i = s_i)$

where X_i is feature, x_i is the value of X_i for a given instance and y is a class label. S_i and s_i respectively denote all features except X_i and its value assignment for a given instance. In other words, a feature is strongly relevant, if removing that feature affects the predictive accuracy.

Weak Relevance
A feature X_i is weakly relevant iff it is not strongly relevant, and there exists a subset of features S'_i of S_i for which there exists some x_i, y and s'_i with $P(X_i = x_i, S'_i = s'_i > 0)$ such that $P(Y = y|X_i = x_i, S'_i = s'_i) \neq P(Y = y|S'_i = s'_i)$

Weak relevance indicates that a feature does not always contribute to predictive accuracy. Such a feature is usually correlated with one or more other features. For instance in a boolean case where F_1 is relevant and $F_2 = \overline{F_3}$, F_2 and F_3 are weakly relevant. If we leave F_2 out, F_3 becomes strongly relevant and vice versa. Another way of saying this is that F_2 and F_3 are redundant, because knowing either one of them implies also knowing the other one. Finding redundant features is often more difficult, because many features are not completely correlated as in this case.

If features are neither strongly relevant nor weakly relevant, they are irrelevant. Irrelevant features can not contribute to predictive accuracy and therefore can always be removed. The challenge it to remove all irrelevant and redundant features, giving you the optimal set as depicted in figure 3.1.

Irrelevance
A feature X_i is irrelevant iff
$\forall\ S'_i \subseteq S_i,\ P(Y = y|X_i = x_i, S'_i) = P(Y = y|S'_i)$

Although these three definitions are sufficient to put features in three distinct categories, Blum and Langley [4] add *relevance as a complexity measure* and *incremental usefulness*. These measures are pretty intuitive and can be defined from the above definitions. Because many algorithms use these measures, I will briefly discuss them.

Relevance as a complexity measure
Given a sample of data S and a set of concepts C, let r(S, C) be the number of features strongly relevant to a concept in C that, out of all those whose error over S is least, has the fewest relevant features.

In other words, we try to find a minimal subset of features for the best classifier for a given concept C.

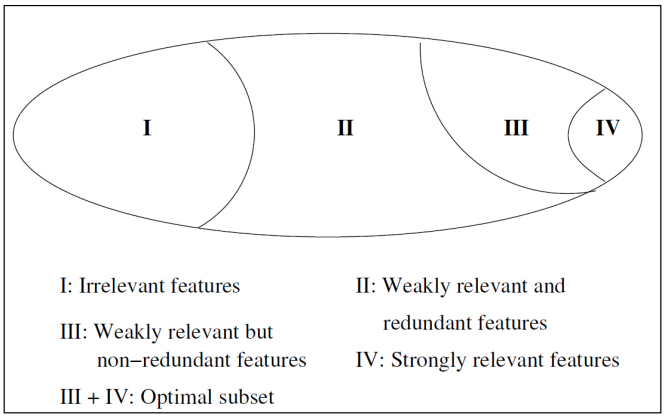


Figure 3.1: Feature relevancy and redundancy, depicted by [32]

Incremental usefulness
Given a sample of data S, a learning algorithm L, and a feature set A (where $x_i \notin A$), feature x_i is incrementally useful to L with respect to A if the accuracy of the hypothesis that L produces using the feature set $x_i \cup A$ is better than the accuracy achieved using just the feature set A.

So in this definition, we look at the predictive accuracy of L, by evaluating it after adding or removing a feature. This kind of measure is at the heart of sequential forward selection (SFS) and sequential backward elimination (SBE). I'll explain more about these methods in chapter 5 on Relief-based algorithms.

3.3 Information and dependency measures

3.3.1 Entropy and information gain

In supervised learning tasks, it's likely that the value of a specific feature contributes to the classification task. Relevant features will contribute most to this classification task. So if we know the classes, we can in principle 'reverse engineer' what features are relevant. One of the methods that uses this principle is information gain, which I will illustrate using a simple sunburn example from [20]. Information gain measures the amount of uncertainty reduced by knowing a feature. Uncertainty is usually measured using Shannon's entropy.

Entropy

Shannon's entropy is a measure for the amount of uncertainty in a distribution. For instance if you have a coin and you flip it, there are two possible outcomes. If it's a fair coin, it's difficult to predict the outcome. If the

Id	Hair	Height	Weight	Lotion	Result
1	blond	average	light	no	sunburned
2	blond	tall	average	yes	none
3	brown	short	average	yes	none
4	blond	short	average	no	sunburned
5	red	average	heavy	no	sunburned
6	brown	tall	heavy	no	none
7	brown	average	heavy	no	none
8	blond	short	light	yes	none

Table 3.1: Sunburn dataset

coin is biased, it's easier to predict the outcome. Shannon's entropy $E(S)$ is given by:

$$E(S) = -\sum_{i=1}^{n} P(i) \log_2 P(i), \tag{3.1}$$

Where n is the number of possible outcomes and $P(i)$ is the probability for outcome i. In this formula you can see that highest entropy is given to uniform distributions, which makes sense, because if all chances of outcomes are equal, it's more difficult to predict the outcome.

Information Gain

Now if we could take away some of the uncertainty, we could make a better prediction. We can do this by splitting up our dataset for every feature, and compare the entropy before the split and after the split by feature X. If we use Shannon's entropy on our dataset, we get the information for all data with d number of classes.

$$Info(D) = -\sum_{i=1}^{d} P_d(c_i) \log_2 P_d(c_i), \tag{3.2}$$

where $P_d(c_i)$ is the observed probability for each class P_d. Now we split the data D by feature X into p parts $\{D_1, D_2, .., D_p\}$. The information for part D_j is:

$$Info(D_j^X) = -\sum_{i=1}^{d} P_{D_j^X}(c_i) \log_2 P_{D_j^X}(c_i), \tag{3.3}$$

with $Info(D_j^X) = 0$ if p_i equals zero in the limit $\lim_{p_i \to 0} \log_2 p_i = 0$. The information gain due to feature X is:

$$InfoGain(X) = Info(D) - \sum_{j=1}^{p} \frac{|D_j|}{|D|} Info(D_j^X), \tag{3.4}$$

where $|D|$ and $|D_j|$ is the number of instances in D and D_j respectively.

Consider the feature *Hair* in table 3.1. *Hair* can have three distinct values, so we can split our dataset for $feature_{hair}$ in three datasets.

Nr	Result
1	SB
2	-
4	SB
8	-

(a) Blond (bl)

Nr	Result
3	-
6	-
7	-

(b) Brown (br)

Nr	Result
5	SB

(c) Red (r)

Table 3.2: Correlation between hair color and sunburn

Now we can calculate the information gain by filling in the numbers from table 3.2 into the equations. The results are depicted in table 3.3.

$\text{Info}(D) =$	$-\frac{5}{8}\log_2\frac{5}{8} - \frac{3}{8}\log_2\frac{3}{8} \approx 0.95$
$\text{Info}(D_{bl}^{Hair}) =$	$2 \times (-\frac{1}{2}\log_2\frac{1}{2}) = 1$
$\text{Info}(D_{br}^{Hair}) =$	0
$\text{Info}(D_{r}^{Hair}) =$	0
$\text{InfoGain}(Hair) =$	$0.95 - \frac{4}{8} \times 1 \approx 0.45$

Table 3.3: Information Gain example

We can also do this for the other features and order them from highest gain to lowest. The features with highest gain are the most relevant. In this case $\{hair, lotion, height, weight\}$. Information gain tends to favor features with more distinct values. Consider equation 3.4; more distinct features do not affect the first part of the equation $Info(D)$. It does however affect the second part. Consider the extreme where we take *Id* as a feature. In this case every class contains exactly one example, having $Info(D_{Id}) = 0$ for every Id, thereby maximizing the information gain. In other words, Id is very good predictor for the class, but is useless for generalization. Quinlan [26] suggested to normalize information gain by a measure called *split information*, which is basically the entropy of the feature in the complete dataset.

$$SplitInfo(X) = -\sum_{i=1}^{n} \frac{|T_i|}{|T|} \times \log_2\left(\frac{|T_i|}{|T|}\right), \tag{3.5}$$

where T is the dataset, partitioned into datasets T_i by feature X. The normalized information gain measure is called *information gain ratio*:

$$GainRatio(X) = \frac{InfoGain(X)}{SplitInfo(X)} \tag{3.6}$$

The gain ratio tries to maximize equation 3.6, with the constraint that information gain should also be high, to prevent choosing features with only low split information.

3.3.2 Mutual information

Another method is to measure the dependency between a feature and a class. Mutual information is such a method:

$$MutualInfo(X_i; C) = \sum_{c \in C} \sum_{x_i \in X_i} P(x_i, y) \log \frac{P(x_i, y)}{P(x_i)P(y)}, \tag{3.7}$$

where C is the set of classes and X_i is the set of features at position i. High values of mutual information reflect high dependency between a feature and a class. In the case of independence, mutual information = 0. Mutual information can also be explained as the amount of uncertainty in a feature removed by knowing its class:

$$MutualInfo(X_i; C) = H(X_i) - H(X_i|C), \tag{3.8}$$

where $H(X_i)$ is the entropy in X_i and $H(X_i|C)$ is the entropy of X_i after knowing C. So if knowing C provides much information, $H(X_i|C)$ will be low and therefore *MutualInfo* will be high. Mutual information is widely used in feature selection, and also in sequence alignments. Usually you select the top k features, with highest mutual information. For instance Peng, Long and Ding [24] use it in their algorithm *minimal-redundancy-maximal-relevance*. One drawback of mutual information is that it prefers rare features in case that the conditional probabilities $P(X_i|C)$ are equal [30]. This may not always be a desired result.

3.3.3 Chi-square (χ^2)

Chi-square is a very simple measure for the difference between observed and expected frequencies. Due to its simplicity and effectiveness, it is widely used in statistics and machine learning. In the case of feature selection, it can be stated as:

$$X^2(x_i) = \sum_{c \in C} \frac{e(c, x_i) - o(c, x_i)}{e(c, x_i)}, \tag{3.9}$$

where the expected number of feature x_i in C is computed as:

$$e(c, x_i) = n_c \frac{a_{x^i}}{N},$$

Where n_c is the number of proteins in subfamily c, N is the total number of proteins and a_{x^i} is the number of proteins having feature x at position i. It's easy to see that when expected and observed frequencies are equal, chi-square is zero, reflecting their independence in the observed data.

In [8] the authors use χ^2 in a multivariate feature selection method for the simple Naive Bayes and Decision Tree classifiers on the GPCR dataset. They use *n-grams* of n subsequent features and use chi-square to select the top K *n-grams* with highest chi-square value. A nice outcome of their research was that Naive Bayesian Classifiers and Decision Trees, the simplest classifiers outperformed state-of-the-art algorithms like SVM in classifying sequences. This is probably mainly due to the effective feature selection process they performed prior to the classification task.

Chi-square and information gain prove to be reliable and useful in feature selection for text categorization [30]. Feature selection in text classification and bioinformatics prove to be very similar, and therefore these methods have also been adopted in various algorithms for feature selection in bioinformatics.

3.4 Wrapper and filter approach

Feature selection algorithms can be divided into two basic approaches [18]. First is the *wrapper approach*, where the selection of features is 'wrapped' within a learning algorithm. The second method is called the *filter approach*, where the features are selected according to data intrinsic values, such as information, dependency or consistency measures.

The difference between these approaches is in the way the 'quality' of the feature subset is measured. For wrapper approaches, a common method is to measure the predictive accuracy for a given subset. Then, applying some forward or backward selection method, the subset is changed and the accuracy is re-evaluated. If it's better, the new subset is retained, otherwise the old one is kept. You can iterate this until you reach a given number of features, some accuracy threshold, after a fixed number of iterations, or when you've explored the whole search space. For this method, the learning algorithm has to run for each subset, so it should not be too demanding. This is the main drawback of this approach.

The other method is the *filter method*. As mentioned, this method uses the intrinsic properties of the data and therefore is not dependent on the learning algorithm. Information, dependency or consistency measures are usually less complex (in time). Therefore these algorithms can be used on large datasets, to reduce the input dimensionality. After that, you can use a classifier or a wrapper to deal with the reduced dataset.

In bioinformatics, datasets are often very large. Therefore the filter approach is mostly used to select the features. Another advantage of this

approach is that you can use any classifier to evaluate the accuracy of the testset. In the wrapper approach, if we use a different classifier for the testset and trainingset, results may be negatively affected.

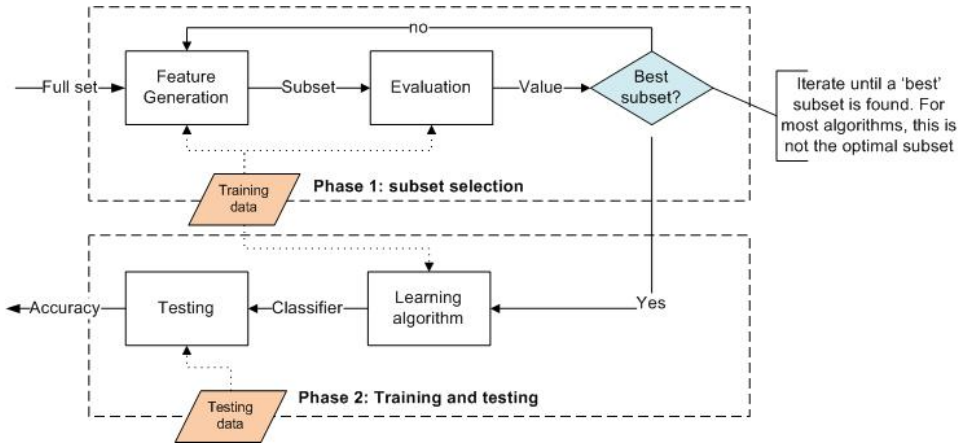


Figure 3.2: Wrapper and filter method in a unified model

Figure 3.2 shows a unified model for these these two approaches from [20]. The difference is in the evaluation method, where filter approaches use a evaluation measure independent of the learning algorithm. In the wrapper approach, the evaluation is performed by the learning algorithm itself, taking the predictive accuracy as a measure for feature quality.

3.5 Univariate and multivariate feature selection

The difference between univariate and multivariate selection is in considering the relationship between features. If you select features by adding, deleting or comparing one feature at a time, this is called univariate feature selection. It is similar to the bag-of-words assumption discussed earlier, assuming independence of features.

One of the main drawbacks of univariate methods is that they do not model dependencies between features. On the other hand, multivariate models often suffer from an explosion of possible feature sets, looking at n features simultaneously, increases complexity by the power of n.

Chapter 4

Related work

Due to the explosion of data in bioinformatics, many feature selection algorithms have been developed. Some are based on evolutionary trees (in biology not only species, but also proteins have a sort of evolutionary tree). Some are based on chemical principles such as hydrophobicity, charge and polarity, others can incorporate 3d structure (although very little is known) or other biological principles. Many are like Relief-based algorithms, based on multiple sequence alignments. I will discuss some of the most successful and known methods for MSA's, because these can be best compared with the Relief-based methods discussed in the next chapter.

Many algorithms based on information theory exist, and it would be impossible to discuss them all. I have selected the most successful and known algorithms. A second reason for discussing these algorithms is that the area under (AUC) the ROC-curve values on many datasets have been published. It is interesting to compare these results with those from my own experiments and try to explain their differences. This is done in chapter 7.

4.1 SDP-Pred

Specificity-determining positions prediction (SDP-pred) [14] is a tool based on mutual information. It uses the statistical association between the value of an amino-acid α and a position i in a MSA as two discrete random variables.

$$I_p = \sum_{i=1}^{N} \sum_{\alpha=1}^{20} f_p(\alpha, i) \log \frac{f_p(\alpha, i)}{f_p(\alpha) f(i)'}, \tag{4.1}$$

where $f_p(\alpha, i)$ is the fraction of residues at position p having amino-acid α in subfamily i, $f_p(\alpha)$ is the frequency of residue α in the whole alignment column, $f(i)$ is the fraction of proteins belonging to subfamily i.

To handle small sample size and biased composition, the amino-acid frequencies are smoothened using a substitution matrix. After that statis-

tical significance is computed based on random shuffling. The important residues are returned based on cut-off for which the threshold is computed by a Bernoulli estimator.

4.2 Sequence Harmony

Sequence Harmony [25] is a relative entropy based method for feature selection. It uses a derivation of Shannon's entropy for biological sequences:

$$rE_i^{A/B} = \sum_x p_{i,x}^A \log \frac{p_{i,x}^A}{p_{i,x}^B} \tag{4.2}$$

where $p_{i,x}^A$ and $p_{i,x}^B$ is the probability of amino-acid type x being observed at position i in family A and B respectively. For an amino-acid to be of maximum importance, it should be present in family A and absent in B or vice versa. Using equation 4.2, this gives a unwanted result, so the authors have introduced sequence harmony:

$$SH_i^{A/B} = \sum_x p_{i,x}^A \log \frac{p_{i,x}^A}{p_{i,x}^A + p_{i,x}^B} \tag{4.3}$$

In general $SH^{A/B} \neq SH^{B/A}$. So to find a 'weight' for position i, the average is taken, yielding the final sequence harmony formula:

$$SH_i = \frac{1}{2}(SH_i^{A/B} + SH_i^{B/A}) \tag{4.4}$$

The method is called sequence harmony, because it looks at the 'harmony' of two subfamilies at a given position. If they have all different amino-acids at that position, the harmony is 0. Identical distributions have maximal harmony with value 1. So important sites have *low* harmony.

4.3 Xdet

Xdet [23] is a method that uses the functional classification of a protein to find specificity determining residues. It does this by correlating two matrices. The first matrix contains the amino-acids changes for two proteins i and j at a given position k (for instance BLOSUM [1] can be used). The second matrix contains the functional similarity between the corresponding proteins. If no quantified similarity information is known, 0 can be used for different and 1 for similar proteins.

After the construction of these two matrices, specificity determining residues can be found using a Spearman rank-order correlation coefficient.

[1] http://en.wikipedia.org/wiki/BLOSUM

$$r_k = \frac{\sum_{i,j}(A'_{ijk} - \bar{A}') \cdot (F'_{ij} - \bar{F}')}{\sqrt{\sum_{i,j}(A'_{ijk} - \bar{A}')^2} \cdot \sqrt{\sum_{i,j}(F'_{ij} - \bar{F}')^2}}, \tag{4.5}$$

where A_{ijk} is the similarity between the amino acids of proteins i and j at position k. F_{ij} is the functional similarity between these proteins and A' and F' are the ranked values of A and F. $\bar{A}$ and $\bar{F}$ are the average values of the ranked matrices. The rank r_k is thus a measure of importance of a given residue k, where higher values correspond to more important features.

The most important property of $Xdet$ is thus that it doesn't use a measure of the sequence hierarchy as a subgrouping property. Instead it uses the functional classification.

4.4 Protein-Keys

This method finds functional residues and subfamilies by employing a combinatorial entropy optimization on a given MSA. The intuition behind this algorithm can be described as follows. Divide a MSA into subfamilies such that each subfamily has a characteristic conservation at some residue positions. Then optimize this information by achieving a compromise between the number of conserved residues and the number of subfamilies. At the two extremes, you can have one subfamily containing all residues, or one protein per subfamily. Both give no information, thus the optimization is somewhere in between. To solve this problem the authors introduce a measure to compare the grouping of sequences into subfamilies, a notion of what is the 'best' distribution and an optimization function to solve this problem.

Measuring by combinatorial entropy

The method uses the general idea that important residues are conserved within subfamilies and are different between them. It uses a simple combinatorial formula to measure the quality of the grouping at position k.

$$Z_{i,k} = \frac{N_k!}{\prod_{\alpha \in [1..21]} N_{\alpha,i,k}!}, \tag{4.6}$$

Where Zi, k is the number of permutations of position i in subfamily k. N_k is the number of sequences in subfamily k, $N_{a,i,k}$ is the number of amino-acids of type α in subfamily k, where gaps are treated as the 21th residue. The values for each position are treated as independent and thus can be summed to obtain the quality of the subgroupings.

$$S = \sum_i \sum_k lnZ_{i,k} \tag{4.7}$$

It is not difficult to see that the entropy is equal to zero if all proteins are put in different subfamilies and maximal if just one subfamily is used.

Best residues

To optimize the method, the authors had to define what 'best' is. The best grouping is where there are as much conserved amino-acids as possible. For instance if a position k is fully conserved within a subfamily, the entropy at that position is 0. If you would have a random grouping of amino-acids, the residue frequencies would follow a uniform distribution, yielding maximum entropy.

Optimization

The optimization method measures the conditional entropy S for a given grouping of subfamilies. It compares this to a measure where the amino-acids are uniformly distributed $\tilde{S}$, which is computed very similar to 4.6. The optimization is thus defined as:

$$\Delta S_i = \left| S_i - \tilde{S}_i \right|, \tag{4.8}$$

Where $|\cdot|$ is the absolute operator. The optimal solution thus is the largest delta between the observed conditional entropy, and the maximum conditional entropy.

Due to the combinatorial explosion for even a small number of proteins of short length, this algorithm can not perform a full search. The authors use a deterministic hierarchical clustering. A nice property of this algorithm is that it is unsupervised and returns also the subgrouping optimizing the found functional specific residues. This can be helpful if the subfamilies are unknown.

4.5 PROUST-II

PROUST-II [12] is a method that uses hidden Markov models and cumulative relative entropy to find relevant residues. From a given MSA A, with subfamilies $S_1, S_2, \ldots, S_k$ the subalignment from A corresponding to S_j is taken. From this subalignment A_j a hidden Markov model is build using an external webserver, resulting in a profile P^j. The profile is converted into a probability profile such that for every amino acid x at position i, the following holds:

$$\sum_x p^j_{i,x} = 1 \tag{4.9}$$

Let $\bar{s}$ denote all subtypes except s. Now, the role of the alignment position i in determining the subtype S_j can be computed using relative entropy.

$$RE_i^s = \sum_x P_{i,x}^s \log \frac{P_{i,x}^s}{P_{i,x}^{\bar{s}}} \tag{4.10}$$

To find the role of an alignment position in determining the sub-types, we have to sum over all subtypes.

$$CRE_i = \sum_s RE_i^s \tag{4.11}$$

The last step is to convert these cumulative values into Z-scores, to find significant residues.

$$Z_i = \frac{CRE_i - \mu}{\sigma} \tag{4.12}$$

Experiments have shown that residues with a Z score > 3.0 are believed to determine specificity.

4.6 SPEER

Specificity prediction using amino-acid properties, entropy and evolution rate (SPEER) [7] is a new method that combines three approaches for the feature selection problem. It uses relative entropy on MSA's, Eucledian distance on physico-chemical properties and computes maximum likelihood on phylogenetic trees to incorporate evolution rate.

These three measures are not new, but combined they provide complementary information and the authors have proven that this information gives better results.

4.7 Multi-Harmony

The authors of Sequence Harmony (SH) have combined their method with an adapted approach of Multi-Relief (MR) and named it Multi-Harmony [5]. MR in this case is implemented as an exhaustive deterministic search instead of random sampling. Z-scores for both SH and MR are computed based on random shuffling. These scores can be combined and the webinterface enables the user to adapt the Z-values to tune the false discovery rate. The idea is that residues with a score far away from the mean score are the most interesting.

Chapter 5

Relief-based algorithms

In the previous chapter I've discussed some algorithms based on measures like information gain and chi-square. While these are good algorithms, they have the major drawback of being myopic. That is, they only consider one feature at a time, independent of all the other features. This myopic univariate feature selection is a somewhat naive approach. Usually, the features in a dataset are not independent. In the case of proteins, this is certainly not the case, as amino-acids interact giving the protein its unique folded structure that determines a large part of its functionality.

To mitigate this drawback, Kira and Rendell introduced the algorithm Relief in 1994 [16] [17]. It is a supervised learning algorithm that considers global and local feature weighting by first computing the nearest neighbors of a sample. By doing so, it takes the whole feature space into account, before updating the relevance for a feature. The intuition is that if you try to find the important properties of an object X in a class, you look at an object Y very similar to X in the same class and one Z in a different class. Then the features that are the same in X and Y, but different between X and Z are probably important. I'll explain this intuition in more detail using an example of cars and boats, in example 5.2.1.

This chapter is constructed in the following way. First I'll illustrate the principles of the original Relief algorithm. I will discuss the major drawbacks and some extensions made to solve them. Relief still proves to be a good basis for new state-of-the-art algorithms. In the subsequent section I will discuss some of the mathematical fundamentals of Relief and show why this is such an effective and efficient tool. This chapter finishes with an explanation of Multi-Relief, which is designed especially for MSA data. The knowledge about these algorithms is essential for the next chapter. In that chapter the general notion of within-class conservation and inter-class divergence is slightly adapted. But first let's take a closer look at why Relief is such an effective algorithm.

5.1 The myopic drawback of impurity functions

The majority of feature selection and evaluation functions are based on impurity measures. Entropy, information gain and Gini-index are good examples. The problem with these measures is that they are univariate and thus myopic, in the sense that they can only look at one feature at a time. They can not take the context of other features into account and are therefore not appropriate for problems with a high degree of feature interaction. This problem is illustrated in [20], which I have also depicted in figure 5.1

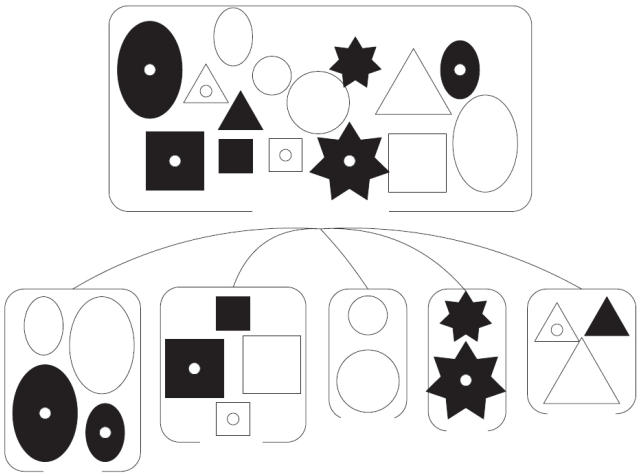


Figure 5.1: Feature selection based on impurity measures. The method is myopic and can not make a distinction between more than one feature at a time.

In this figure the geometrical objects are instances and their shape reflects a feature. The color (black or white) defines the class. The figure illustrates that impurity function splits the data by one feature at a time. It can separate the ellipse, square, circle etcetera. After this separation, it can not separate any further based on the containment of circle or the size. Hence there was a need for a simple, fast algorithm that also takes the whole feature space into account and reliefs us from the myopic drawbacks of only looking at local differences. Its probably not hard to guess the name of this algorithm ...

5.2 The original Relief-algorithm

Relief was proposed by Kira and Rendell in 1994 [16]. The success of the algorithm is due to the fact that it's fast, easy to understand and implement and accurate even with dependent features and noisy data. The algorithm is

based on a simple principle. We like to put objects with similar properties in a class. Some of these properties (or features) are very important in the classification task and others are less important. It's probably best to illustrate this with a simple example of separating boats from cars.

Example 5.2.1 The intuition behind Relief
Consider that we have many boats and cars. Both classes have some well-known properties in common. For instance you can travel with them, they have an engine, a steering wheel etcetera. You can say that not every boat has an engine, some are sailing boats and hence rely on their sail for energy. But if you ask a child what separates boats from cars, they probably say that cars have wheels and boats have not, or boats float on water and cars do not. So these are important features. This is basically what Relief does, if we pick a car, it looks for a car most similar to the one we've picked and also for a boat most similar to the car we've picked. After doing that it looks at the features in which they differ, because these are important features for separating boats from cars.

The same principle can be applied to less obvious examples, such as functional specificity determining residues in MSA's. If we take a sequence X, its nearest neighbor from the same class (the nearest hit NH_X) and nearest neighbor from an opposite class (the nearest miss NM_X), we think the residues or amino-acids that separate X and NM_X the most and are preserved between X and NH_X are the most important. It is a sound assumption that these are also the features that contribute much to the functional specificity of such a residue. The Relief algorithms' pseudo code is depicted in algorithm 1. The algorithm basically consists of three important parts:

1. Calculate the nearest miss and nearest hit;
2. Calculate the weight of a feature;
3. Return a ranked list of features or the top k features according to a given threshold.

The algorithm starts with initializing the weight vector and setting the weight for every feature to 0. After that it randomly picks a learning instance X and calculates the NH_X and NM_X. In the case of numerical data, the obvious distance to use is the *Euclid distance*. In the case of nominal features, as in the case of sequence data the *Hamming distance* is more appropriate.

The Hamming distance between two sequences X_1 and X_2 is the number of positions in which they differ. For instance the Hamming distance between ABCD and ACBD = 2. The nearest neighbor of course is the instance with the smallest distance. After this global feature exploration, the algorithm evaluates the difference between X and its two nearest neighbors

Algorithm 1 RELIEF

Input: M learning instances X described by N features; T iterations
Output: for each feature F_i a quality weight within $-1 \leq W[i] \leq 1$

1: $\forall i, W[i] = 0$;
2: **for** $t = 1$ **to** T **do**
3: randomly pick an instance X
4: find nearest hit NH_X and nearest miss NM_X of X
5: **for** $i = 0$ **to** N **do**
6: W[i]=W[i] + diff($X^{(i)}$,$\mathrm{NM}_X^{(i)}$)/(M × T) - diff($X^{(i)}$,$\mathrm{NH}_X^{(i)}$)/(M × T)
7: **end for**
8: **end for**
9: **return** (W);

in a local sense; thus one feature at a time. This difference is calculated using equation 5.1.

$$y = \begin{cases} \frac{|x_{j,i} - x_{k,i}|}{max(F_i) - min(F_i)} & F_i \text{ is numerical} \\ 0 & X_{j,i} = X_{k,i} \wedge F_i \text{ is nominal} \\ 1 & X_{j,i} \neq X_{k,i} \wedge F_i \text{ is nominal} \end{cases} \tag{5.1}$$

In Relief, both nominal and numeric features can be used, and they can also be used simultaneously. However in that case, one should be aware that numerical features tend to be underestimated and compensate for this fact. I will not go into this, for I only have to deal with nominal features. Therefore the distance for a feature between two instances is either 0 (they are the same) or 1 (they are different). After each iteration, the weights for a feature are updated and normalized within the interval $[-1, 1]$ by dividing it with the number of samples and iterations. The output is a weight vector, with a weight W_i for each feature i. In the original paper, the authors propose a relevancy threshold τ. If you select all features with a weight $\geq \tau$ you get a subset selection algorithm. The authors describe a statistical mechanism to calculate τ, but for sequence alignments, a ranked list of all features is preferred in most cases. In this case, Relief operates as a feature ranking mechanism.

By now, it should be obvious that the key to the success of Relief lays in the fact that it does a global and a local search. It doesn't rely on greedy heuristics like many other algorithms that often get stuck in local optima. This idea is nicely illustrated in [20] and depicted in figure 5.2. In this picture there are three features {*shape*, *size*, *contains a dot*}. The idea is that a relevant feature can separate two instances from opposite classes that are closely related. Therefore it takes the most closely related instance from an opposite class (the nearest miss) and from the same class (the nearest hit).

Note that in the right hand side, there are two nearest misses, because they both have two of the three features in common with the selected instance.

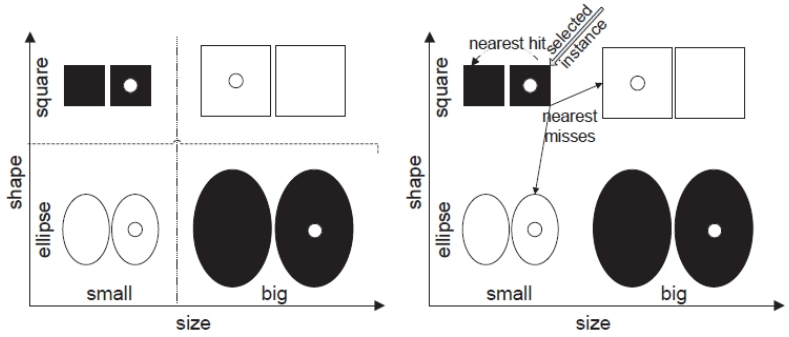


Figure 5.2: Finding nearest neighbors with Relief. The class labels are the colors. Relief chooses the NH and the NM based on the number of features (*shape, size, contains a dot*) they have in common.

5.3 The relation between impurity and Relief

Before going into more detail with the other Relief-based algorithms, it is good to know why Relief is such an effective algorithm and also what are its limitations. There is clear relation between the impurity functions mentioned in chapter 3 and Relief. Let's have a closer look at the background and basis of this successful algorithm. First, I'll give the definition of an impurity function defined by Breiman [6].

Impurity function
An impurity function is a function defined on the set of all K-tuples of numbers $(p_1, ..., p_K)$ *satisfying* $p_j \geq 0$, j = 1,..,K, $\sum_j p_j = 1$ *with the properties:*

1. ϕ *is a maximum only at the point* $(\frac{1}{K}, \frac{1}{K}, .., \frac{1}{K})$
2. ϕ *achieves its minimum only at the points (1, 0, ..., 0), (0, 1, 0, ..., 0), ..., (0, 0, ..., 0, 1).*
3. ϕ *is a symmetric function of* $p_1, ..., p_K$, *i.e., if you permute* p_j, ϕ *remains constant.*

With this definition it is easy to see that entropy is such a function. It has its maximum with a uniform distribution, its minimum if only one of the features differs and it is symmetric. Gini-index gain is another impurity function similar to the entropy based information gain. It is also closely related to Relief. Gini-index gain is the difference between the prior and the expected posterior Gini-indices.

$$Gini(F_i) = \sum_{j=1}^{n_i} p(F_i = j) \sum_{c=1}^{C} p(y = c|F_i = j)^2 - \sum_{c=1}^{C} p(y = c)^2 \tag{5.2}$$

It has been shown by Igor Kononenko [19] that Gini-index gain and Relief are closely related. What Relief tries to approximate is the probability that a feature can separate two classes in a local neighborhood.

$$\begin{aligned} W[F_i] &= \text{P(different value of } F_i|\text{nearest miss)} \\ &- \text{P(different value of } F_i|\text{nearest hit)} \end{aligned} \tag{5.3}$$

When you don't look for just one nearest neighbor, but increase this to all k nearest neighbors, equation 5.3 becomes:

$$\begin{aligned} W[F_i] &= \text{P(different value of } F_i|\text{different class)} \\ &- \text{P(different value of } F_i|\text{same class)} \end{aligned} \tag{5.4}$$

I'll use a shorter notation: P_{ev} = P(equal value of F_i), P_{sc} = P(same class), $P_{sc|ev}$ = P(same class| equal value of F_i). Now we can rewrite 5.4 using Bayes rule:

$$W[F_i] = \frac{P_{sc|ev}P_{ev}}{P_{sc}} - \frac{(1 - P_{sc|ev})P_{ev}}{1 - P_{sc}} \tag{5.5}$$

Where the prior class probability is defined as:

$$P_{sc} = \sum_{c \in classes} P(c)^2 \tag{5.6}$$

Equation 5.6 is common sense, because the probability that two instances are in a given class is the probability of one instance being in that class times the probability of the other being in that class. Summing over all classes gives the probability of two instances being in the same class. We can also obtain $P(sc|ev)$, as demonstrated by Kononenko. I will use a derivation similar from [15], where same class is represented as $c_1 = c_2$. Similarly, equal value is represented by $v_1 = v_2$.

$$\begin{aligned} P_{sc|ev} &= \frac{P(c_1 = c_2 \cap v_1 = v_2)}{P(v_1 = v_2)} \\ &= \sum_v \frac{P(c_1 = c_2 \cap v_x = v_y)}{\sum_v P(v)^2} \\ &= \sum_v \frac{P(c_1 = c_2|v_x = v_y)P(v)^2}{\sum_v P(v)^2} \\ &= \sum_v \sum_c \frac{P(c|v)^2)P(v)^2}{\sum_v P(v)^2} \\ &= \sum_v \frac{P(v)^2}{\sum_v P(v)^2} \sum_c P(c|v)^2 \end{aligned} \tag{5.7}$$

Substituting 5.6 and 5.7 in equation 5.5, we get:

$$\begin{aligned} W[F_i] &= \frac{(1-P_{sc})P_{sc|ev}P_{ev}}{P_{sc}(1-P_{sc})} - \frac{P_{sc}(1-P_{sc|ev})P_{ev}}{(1-P_{sc})P_{sc}} \\ &= \frac{P_{sc|ev}P_{ev}}{P_{sc}(1-P_{sc})} - \frac{P_{sc}}{(1-P_{sc})P_{sc}} \\ &= \frac{P_{ev} \times Gini'(F_i)}{P_{sc}(1-P_{sc})} \end{aligned}$$

Where $Gini'(F_i) = P_{sc|ev} - P_{sc}$. The prior probability of a class is not affected by the attribute and thus a constant. We can rewrite this as:

$$Constant \times \sum_v P(v)Gini'(F_i) \tag{5.8}$$

Now the only difference between the Gini-index gain and $Gini'$ is that the latter uses the factor $\frac{p(v)^2}{\sum_v p(v)^2}$, where the first uses $\frac{p(v)}{\sum_v p(v)} = p(v)$. It's known that impurity functions tend to overestimate features with a larger number of distinct values. By using the factor $\sum_v p(v)^2$ in 5.8, there is a sort of implicit normalization for this effect. Thus while impurity functions overestimate multi-valued attributes, Relief has no such undesired property.

The derivation above holds for a larger number of nearest neighbors. By taking more nearest neighbors into account, you get the algorithm known as *Relief-A*. By using more neighbors, *Relief-A* can handle some noise. However, in many applications, even a 1-NN classifier yields good results and 'upgrading' this to *K-NN* gives only modest improvements.

5.3.1 Missing values in data

Relief-A can cope with noisy and redundant features, but has no mechanism to handle missing attributes. Kononenko discussed three possible solutions (Relief B, C and D) of which *Relief-D* performed significantly better than the other two. It can handle missing data in cases that only one or both instances have a missing value for the given attribute. It does this by extending the difference measure with these two cases:

- if one instance has a missing value for the given attribute (for instance a 'gap' in a MSA):

$$\text{diff}(F_i, X_l, X_k) = 1 - P(F_i = X_{k,i} | y = y_l) \tag{5.9}$$

In this case look at feature i of the instances we do know and which are in the same class as X_l. We can compute the probability that they are the

same. By subtracting this probability from 1, we get the probability that they are different as an estimation of the probability that X_l and X_k are different also.

- if both instances have a missing value for the given attribute:

$$\text{diff}(F_i, x_l, x_k) = 1 - \sum_{j=1}^{n_i}(P(F_i = j|y = y_l) \times P(F_i = j|y = y_k)) \quad (5.10)$$

If both are different the same applies. This time we look at the possibility that any two features from the two classes are the same. By summing over all possibilities and subtracting this from 1, we receive the desired estimate.

5.3.2 Multi-class data

The original algorithm can handle only two-class problems. Kononenko proposed two extensions of Relief, which can handle multi-class problems. The first is a straight-forward extension, where the nearest hit calculation stays the same and one nearest miss is calculated for all classes. This method (*Relief-E*) seems unsatisfactory. A better method is known as *Relief-F*, where the nearest miss is calculated for every opposite class (that is every class different from the class of the instance X_l picked). The result is weighted with the prior probability of each opposite class and averaged over all opposite classes. The new weight update function thus is:

$$W[F_i] = W[F_i] - \frac{\text{diff}(F_i, x_l, NH_{x_l})}{m} + \sum_{c \neq class(x_l)} \frac{[P(C) \times \text{diff}(F_i, x_l, NM(C))]}{m}$$

Where NM and NH respectively denote the nearest hit and nearest miss. The idea is that *Relief-F* measures not only the ability of an attribute to separate it from another class, but the ability that an attribute separates any pair of classes, no matter how close they are.

5.4 RELIEF-F

The resulting algorithm of the previous section is known as *Relief-F*. Because this is such an important algorithm, the pseudo code is given in algorithm 2.

The algorithm looks a lot like Relief (algorithm 1), but incorporates three important improvements. First, it is less sensitive to noise, because it looks for n nearest instances (line 6 and 9). Second, it includes the above mentioned strategy for coping with missing values (line 11, 13). And last but not least, the algorithm can handle multi-class data (line 6) and normalizes the weights with the a priori probability of the class $\frac{p_y}{1-p_{y_k}}$ at line 13.

Algorithm 2 RELIEF-F

Input: M learning instances X_k described by N features; C classes; m iterations; class probability p_y; number of n nearest instances from each class

Output: for each feature F_i a quality weight within $-1 \leq W[i] \leq 1$

for $i = 0$ **to** N **do**
  W[i]=0.0
end for
for $l = 1$ **to** m **do**
  randomly pick an instance X_k (with class y_k);
  for $y = 1$ **to** C **do**
    find n nearest instances $x[j, y]$ from class y, where $j = 1..n$;
    for $i = 1$ **to** N **do**
      for $j = 1$ **to** n **do**
        if $y = y_k$ **then** {nearest hit}
          W[i] = W[i] + diff$(i, x_k, x[j, y])/(m \times n)$;
        else {nearest misses}
          W[i] = W[i] + $p_y/1 - p_{y_k} \times$ diff$(i, x_k, x[j, y])/(m \times n)$;
        end if
      end for
    end for
  end for
end for
return (W);

5.5 Margin based Feature selection

In [9], the authors introduce the idea of margin based feature selection. *A margin is a geometric measure for evaluating the confidence of a classifier with respect to its decision.* Margins are not new to machine learning, they are for instance widely used in support vector machines. Margins are used for theoretic bounds and can be used for the design of new algorithms.

There are two types of margins:

- *sample margin*; the sample margin measures the distance between the instance and the decision boundary induced by the classifier. (for example used in SVM).
- *hypotheses margin*; the hypothesis margin with respect to an instance is the distance between the hypothesis and the closest hypothesis that assigns alternative label to the given instance.

Hypothesis-margin

Let P be a set of points and x be an instance. Let w be a weight vector over the feature set, then the margin of x is:

$$\theta_p^w = \frac{1}{2}(|x - NM(x)|_w - |x - NH(x)|_w) \tag{5.11}$$

$$where\ |z|_w = \sqrt{\sum_i w_i^2 z_i^2}$$

It is natural to normalize such that $max\ w_i^2 = 1$, because this guarantees that $|z|_w \leq |z|$, where $|z|$ represent the Euclidean norm. In order to compare two feature sets, we need an evaluation function. The evaluation is the sum of all the margins in the sample.

Evaluation function
Given a training set S and a weight vector w, the evaluation function is defined as

$$e(w) = \sum_{x \in S} \theta_{S \setminus x}^w(x) \tag{5.12}$$

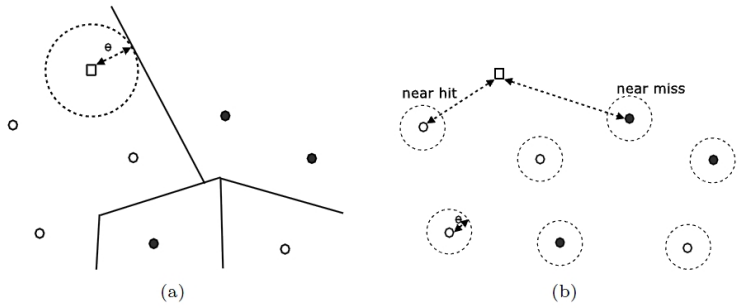


Figure 5.3: Sample margin (a) and hypothesis margin (b) for K-NN

Figure 5.3 [11] shows the two types of margins for the nearest neighbor classifier. The circles reflect the sample points, the square is a new instance x. The sample margin (a) is the distance between the boundary and the instance. The hypothesis margin (b) is the maximum distance the *sample points can travel* without altering the label of the new instance x. In this case it is $\frac{1}{2}(|x - NM(x)| - |x - NH(x)|)$. It has been shown that the sample margin for 1-NN can be unstable, and thus the hypothesis margin is preferred. Two important points are:

1. The hypothesis-margin of an instance X with respect to a set of points P can be easily obtained using: $\theta_p(x) = \frac{1}{2}(|x - NearestMiss(x)| - |x - NearestHit(x)|)$.
2. The hypothesis-margin lower bounds the sample margin.

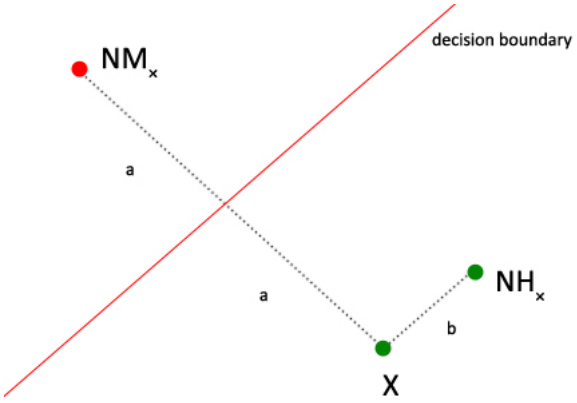


Figure 5.4: Relation between hypothesis- and sample margin

The two important points stated above are shown in figure 5.4. The sample margin a is illustrated as the distance of X to the decision boundary. Following point 1, the hypothesis margin is $\frac{1}{2}(2a - b)$. Since b is a distance, it holds that $b \geq 0$, and thus $\frac{1}{2}(2a - b) \leq a$, which shows point 2.

It's important to note that a chosen set of features influences the nearest hit and nearest miss, just as in original Relief. Therefore, the chosen set of features directly influences the hypothesis-margin, through the distance measure. With the above definition of a margin, it's a logical step to look for a set of features that maximizes this margin and thereby the confidence in the classification task. This is exactly the intuition behind the algorithms *G-Flip* and *Simba* introduced by the authors. G-flip is a greedy heuristic consisting of four basic (a little simplified) steps:

1. Start with an empty set of features $F = \emptyset$;
2. Calculate the average margin between the sample instances given the feature set F and a weight vector w;
3. Evaluate $(F + F_i)$ vs $(F - F_i)$, for a randomly selected feature F_i and keep the set that maximizes the margin;
4. Iterate these three steps until convergence.

The margin is calculated based on the definition of the hypothesis-margin above. The authors re-evaluate the weight vector by taking the one that maximizes the average margin at each iteration (where $w_i = 1$ for $F_i \in F$ and 0 otherwise). The algorithm is based on a greedy heuristic and sensitive to local optima. The authors try to mitigate this effect, by restarting each iteration with a random permutation of the features to flip. Note that this algorithm incorporates the sequential forward selection method discussed in chapter 3.

5.5.1 G-flip

Algorithm 3 Greedy feature flip (G-flip)

Input: M learning instances, described by N features
Output: set of relevant features F

```
1: initialize the set of relevant features F = ∅
2: for t = 1 to T do
3:    pick a random permutation s of 1..N
4:    for i = 1 to N do
5:       evaluate e1 = e(F ∪ s(i)) and e2 = e(F \ s(i))
6:       if e1 > e2 then {margin is bigger with feature s(i)}
7:          F = F ∪ s(i)
8:       else if e2 > e1 then {margin is bigger without feature s(i)}
9:          F = F \ s(i))
10:      else {no change}
11:         break
12:      end if
13:   end for
14: end for
15: return (F);
```

G-flip is a greedy search algorithm that tries to maximize the evaluation function in equation 5.12. It iteratively adds or removes a feature to the set of features F, by evaluating the margin with and without the feature. It is Monte-Carlo algorithm that converges to a local maximum of the evaluation function. The time complexity is $\Theta(TN^2m^2)$, where T is the number of iterations, N is the number of features and m is the number of instances. The authors claim the algorithm is parameter free, which is true, except for the number of iterations. This is an advantage, it's fast convergence (about 10 to 20 iterations) is another one. However no optimal solution can be guaranteed and the computational complexity is much larger than most other Relief-based algorithms.

5.5.2 Simba

The above algorithm tries to maximize the margin directly. Therefore it must rely on heuristic greedy search, which is often slow and gives no guarantees for an optimum solution. On top of that the weight vector w is often unknown. To overcome these problems, the authors propose an algorithm that computes the weight vector w that maximizes the evaluation function in equation 5.12 directly. They do so by using a stochastic gradient ascent over *e(w)*. The idea is that the algorithm converges, to a feature weight

vector that maximizes the evaluation function.

Algorithm 4 Simba

Input: S learning instances, described by N features
Output: feature weight vector W

1: $\forall i, W_i = 1$
2: **for** $t = 1$ **to** T **do**
3: (a) pick a random instance x from S.
4: (b) calculate *NM(x)* and *NH(x)*, with respect to $S \setminus x$ and weight vector w
5: **for** $i = 1$ **to** N **do**
6: (c) calculate $\Delta_i = \frac{1}{2}\left(\frac{(x_i - NM(x)_i)^2}{|x - NM(x)|_w} - \frac{(x_i - NH(x)_i)^2}{|x - NH(x)|_w}\right) w_i$
7: (d) $w_i = w_i + \Delta_i$
8: **end for**
9: **end for**
10: $w = \frac{w^2}{|w|_\infty}$ where $(w^2)_i = (w_i)^2$
11: **return** (W);

The major advantage of Simba compared to Relief is that is re-evaluates the margin with respect to the updated weight vector. Relief has no such feedback mechanism and is therefore inferior to Simba. They have the same time complexity of $\Theta(TNm)$.

5.6 Optimizing the margin as a convex optimization problem

Relief is a successful algorithm, due to its simplicity and effectiveness. Sun [28] shows that Relief implements an online algorithm that solves a convex optimization problem based on a nearest neighbor margin objective function. Therefore, it performs better compared to most filter methods, because it receives performance feedback from a nonlinear classifier. It's also better than most wrapper methods, because it optimizes a convex problem and therefore avoids exhaustive or heuristic search and thus can be implemented efficiently.

Sun claims two drawbacks of Relief:

1. The nearest neighbors are defined in the original feature space, which are unlikely to be the same as in the weighted feature space.
2. Relief can not deal with outliers.

5.6.1 Optimizing the margin

As described above the hypothesis margin can be defined as:

$$\rho_n = d(X_n - NM(x_n)) - d(X_n - NH(X_n)) \tag{5.13}$$

Where $d(\cdot)$ is a distance function. It seems natural to maximize the average margin in a weighted feature space (which Relief establishes) by scaling each feature:

$$max_w \sum_{n=1}^{N} \rho_n(w) \tag{5.14}$$

When we use the distance metric from 5.13, equation 5.16 becomes:

$$max_w \sum_n \left(\sum_i w_i |x_n^i - NM_{x_n}^i| - \sum_i w_i |x_n^i - NH_{x_n}^i| \right) \tag{5.15}$$
$$\text{subject to } \|w\|_2^2 = 1, w \geq 0$$

where ρ_n is the margin of x_n with respect to w, n is the number of samples and i is the number of features. The constraint $\|w\|_2^2 = 1$ prevents a boundless increasing of the maximization. $w \geq 0$ ensures a distance metric, since a distance can not be negative.

Equation 5.16 can be simplified to:

$$max_w w^T z \tag{5.16}$$
$$\text{subject to } \|w\|_2^2 = 1, w \geq 0,$$

where $z = \sum_n |x_n - NM(x_n)| - |x_n - NH(x_n)|$, and $|\cdot|$ denotes the point-wise absolute operator. The optimum solution is calculated as:

$$w = \frac{(z)+}{\|(z)+\|_2}, \tag{5.17}$$

where $(z)^+ = [max(z_1, 0), max(z_2, 0), \ldots, max(z_I, 0)]^T$.

Basically, the new weights for every position i, is the margin z times the old weight, where distances < 0 are set to 0.

This optimization is incorporated in the update rule of Relief (although the distance is not reweighed at every iteration), which is thus a online solution to this problem. Simba uses this idea for the Eucledian distance, but returns many local optima. This is why it starts every iteration by picking a random instance, and maximizes the margin for this instance.

5.6.2 Iterative-Relief

Basic idea and pseudo code

This algorithm [28] consists of two important steps: the first is to calculate the maximum margin over all sample points given the weight vector. In

the second step, re-estimate the weight vector given this maximum margin. Iterate these steps until the algorithm converges or has passed all iterations. The strength of this algorithm is that it considers the margin over all sample points, thereby avoiding local optima. In the second step, it re-estimates the feature weights in a global sense, thereby trying to increase the margin a little more every iteration. This feedback mechanism is not incorporated in Relief. A secondary strength is a mechanism to take outliers into account, a mechanism that Relief also lacks. Before explaining this algorithm step-by-step, I'll give a description in pseudo code.

Algorithm 5 Iterated Relief

Input: M learning instances, described by J features, T iterations, stop criterion τ

Output: feature weight vector W

1: $\forall\ i,\ W_i = (\frac{1}{\sqrt{N}})$
2: **for** $t = 1$ **to** T **do**
3: **for** $n = 1$ **to** M **do**
4: select instance X_m and calculate $P_o(X_m)$ with respect to $W^{(t-1)}$ like equation 5.20;
5: **for** $i = 1$ **to** M **do**
6: select $i_1 = miss(X_m)$ and $i_2 = hit(X_m)$;
7: calculate $P_m(i_1|X_m, W^{(t-1)})$ and $P_h(i_2|X_m, W^{(t-1)})$ like equation 5.18 and 5.19 respectively;
8: calculate $|X_m - i_1|$ and $|X_m - i_2|$ with respect to $W^{(t-1)}$ and update weight W^t as equation 5.22
9: **end for**
10: **end for**
11: **if** $\|W^t - W^{(t-1)}\| < \tau$ **then**
12: **return** (W^t);
13: **end if**
14: **end for**
15: **return** (W^t);

The working of the algorithm

Consider two sets $M_n = \{1 \leq i \leq N, y_1 \neq Y_n\}$ and $H_n = \{1 \leq i \leq N, y_i = y_n, i \neq n\}$, associated with pattern x_n. We have a set of binary parameters $o = [o_1, o_2, .., o_n]^T$, where $0_n = 0$ if x_n is an outlier and $0_n = 1$ otherwise. Unfortunately, we don't know which are the nearest hits and misses for every pattern x_n, nor do we know whether a pattern is an outlier or not. We do know that we want to optimize the objective function $C(w) = \sum_{n=1,o_n=1}^{N}(\|X_n - \mathrm{NM}_{X_n}\|_w - \|X_n - \mathrm{NH}_{X_n}\|_w)$, which can be optimized using

equation 5.16.

Now a problem is that if we want to calculate the nearest hit and miss for every data point, we have to calculate the distance from X to all other points in S and do this for every $X \in S$. This becomes intractable, so Sun resolved this by assuming all data point are random variables, and calculate the probability that a given data point i is a nearest neighbor. The probability of pattern i being the nearest miss of x_n is given as:

$$\alpha_{i,n} = P_m(i|x_n, w) = \frac{f(\|X_n - X_i\|_w)}{\sum_{j \in M_n} f(\|X_n - X_j\|_w)} \tag{5.18}$$

similar for pattern i being the nearest hit of X_n:

$$\beta_{i,n} = P_h(i|x_n, w) = \frac{f(\|X_n - X_i\|_w)}{\sum_{j \in H_n} f(\|X_n - X_j\|_w)} \tag{5.19}$$

and the probability that x_n is not an outlier is:

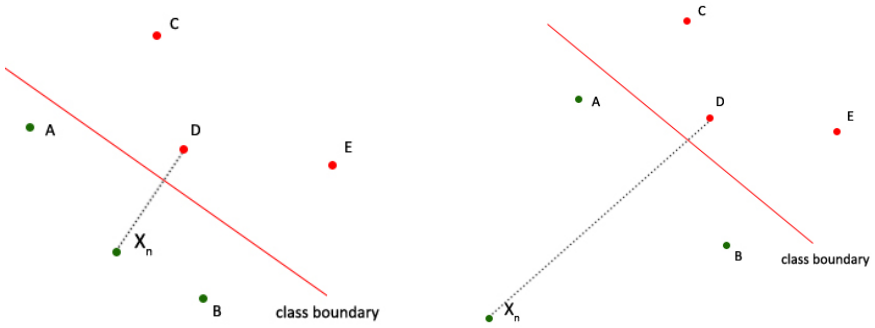
$$\gamma_{i,n} = 1 - P_o(o_n = 0|D, w) = 1 - \frac{\sum_{i \in M_n} f(\|X_n - X_i\|_w)}{\sum_{X_i \in D \setminus X_n} f(\|X_n - X_j\|_w)} \tag{5.20}$$

Where f(·) is a kernel function. In his paper [28], Sun uses the exponential function: $\exp(-d/\sigma)$, where the kernel width σ is a user-defined parameter. Figure 5.5 illustrates the α and γ functions where all weights are equal. It is easy to see that due to the exponential function, a smaller distance yields a higher probability of a nearest miss. The denominator can be seen as a constant for all i. Thus a higher probability of i being the nearest hit is indeed reflected in a closer distance. The probability that i is the nearest hit of X_n can be calculated in the same way, using the $\beta_{i,n}$ function from equation 5.19.

For the outliers, a large distance to the other class samples gives a bigger probability of X_n being an outlier. The equation can be interpreted as $\frac{C}{C+e^{-\Delta\text{Hit}}}$. Where C is a constant reflecting the exponential function of X_n and all the misses. ΔHit is the distance to a hit of X_n. A larger distance thus reflects a bigger probability that X_n is indeed an outlier. The objective function can be maximized using an algorithm strongly related to *expectation maximization* (which works with likelihoods, not distances).

Step (1): Calculating Q after iteration t:

$$Q(w|w^{(t)}) = E_{\{S,o\}}[C(w)] =$$
$$\sum_{i=1}^{N} \gamma_n \left(\sum_{i \in M_n} \alpha \|X_n - X_i\|_w - \sum_{i \in H_n} \beta \|X_n - X_i\|_w \right) =$$



(a) Example where D is the nearest miss of X_n

(b) Example of X_n being an outlier

Figure 5.5: Nearest miss (a) and outlier (b) illustrated

$$\sum_{i=1}^{N} \gamma_n \left(\sum_j w_j \underbrace{\sum_{i \in M_n} \alpha_{i,n} m_{n,i}^j}_{\overline{m}_n^j} - \sum_j w_j \underbrace{\sum_{i \in H_n} \beta_{i,n} h_{n,i}^j}_{\overline{h}_n^j} \right) =$$

$$w^T \sum_{i=1}^{N} \gamma_n (\overline{m}_n - \overline{h}_n) = w^T v \tag{5.21}$$

α, β and γ are defined in equation 5.18, 5.19 and 5.20 respectively. Furthermore, $m_{n,i} = |X_n - X_i|$, for $i \in M_n$ and $h_{n,i} = |X_n - X_i|$ for $i \in H_n$.

Step (2): Setting w for the (t+1)-th iteration:

$$w^{(t+1)} = \arg\max_{w \in W} Q(w|w^{(t)}) = \frac{v^{(+)}}{\|v^{(+)}\|_2} \tag{5.22}$$

These two steps are repeated until $\|w^{(t+1)} - w^{(t)}\| < \tau$.

Multi-class extension The algorithm at this moment can only handle two-class problems. I-Relief can be easily extended with the multi-class definition from Relief-F.

$$w_i = w_i + \sum_{c \in Y, c \neq y(x)} \frac{P(c)}{1 - P(y(x))} |x^{(i)} - \mathrm{NM}_c^{(i)}(x)| - |x^{(i)} - \mathrm{NH}^{(i)}(x)| \tag{5.23}$$

where Y is the set of possible class labels $\{1, 2, .., C\}$, $\mathrm{NM}_c(x)$ is the nearest miss of x from class c and $P(c)$ is the a priori probability for class

c. This equation can also be used to define a sample margin, with one drawback, that a positive margin does not guarantee correct classification. It seems more natural to define the margin as the minimal difference instead of the average weighted difference.

$$\rho = min_{x_i \in D \setminus D_{y(x)}} \left(d(x - x_i) - d(x - \mathrm{NH}(x)) \right) \tag{5.24}$$

It also ensures us that a positive margin corresponds to a correct classification.

Conclusion

In this section I've described how Relief can be formalized as a convex optimization problem. I-Relief is an algorithm that tries to re-estimate the weights of the features be iteratively maximizing the margin as shown in [28]. I-Relief has a time complexity $\Theta = TNM^2$ (T iterations, N feature dimensions, M samples), compared to $\Theta = TNM$ for Relief. Sun also discussed an online algorithm which is based on the same principles as I-Relief, and has the same time complexity as Relief.

5.7 Multi-Relief for sequence alignments

Multi-Relief [31] is designed for determining functional specific residues in MSA's. It does this by repeatedly sub-sampling two randomly selected classes and applying Relief to these sub-samples. Results of these multiple runs are then ensembled.

Algorithm 6 Multi-Relief

Input: C classes of aligned proteins, each protein described by N residues; T iterations; number of S sub-samples
Output: feature weight vector W

1: $\forall\ i,\ W_i = 0$
2: **for** $t = 1$ **to** T **do**
3: select randomly two classes $c_1, c_2 \subseteq C$;
4: X_1, X_2 = select randomly S sample sequences from c_1 and c_2 resp.;
5: W_t = apply Relief(X_1, X_2);
6: **end for**
7: **for** $i = 1$ **to** N **do**
8: $W_i = weight(i)$;
9: **end for**
10: **return** (W);

The weight of a position i is normalized within the boundary $[-1, 1]$ on line 8 by averaging over all the runs that assigned weight to i. Where

$N^+ = |\{W_t(s) > 0 \forall t\}|$ is the average over all the runs that assigned a positive weight and $N^- = |\{W_t(s) < 0 \forall t\}|$ is the average weight over all negative runs. Thus the $weight(i)$ function is defined as:

$$weight(i) = \begin{cases} \frac{1}{N^+} \sum_t \{W_t(i) > 0 \forall t\} & for \quad N^+ > 0 \\ \frac{1}{N^-} \sum_t \{W_t(i) > 0 \forall t\} & for \quad N^+ = 0 \wedge N^- > 0 \\ 0 & for \quad N^+ = N^- = 0 \end{cases} \tag{5.25}$$

In the definition of $weight(i)$ those positions that can separate between at least two classes are given a positive weight. If they can not, but they separate samples within a class, the feature gets a negative weight. If it can not separate anything it gets weight 0. Random sampling pairs of classes is mainly done for efficiency, whereas random sub-sampling is also done to handle unbalanced classes.

5.7.1 Residue relevancy

	a	b	c	d	e
C_1	R	F	T	I	T
	R	F	T	Q	F
	R	F	T	N	V
	R	F	T	A	D
C_2	R	F	Y	S	T
	R	F	Y	F	F
	R	F	Y	D	V
	R	F	Y	L	D
C_3	R	Y	D	E	T
	R	Y	D	V	F
	R	Y	D	W	V
	R	Y	D	G	D
C_4	R	Y	H	H	T
	R	Y	H	P	F
	R	Y	H	Y	V
	R	Y	H	C	D
weights	0	1	1	0	-1

Table 5.1: Weights computed by Multi-Relief applied on a toy example [31]

In table 5.1 there are four subfamilies $\{C_1..C_4\}$, each containing four proteins with five amino-acid positions $\{a..e\}$ each. We can see that Multi-Relief assigns weight = 0 to position a, because it can not separate any subfamily (it is *fully conserved*). Put another way, knowing position a is knowing nothing, because *every* protein has R at position a. The same holds for position d, because every position in d is different and thus d doesn't tell anything about a subfamily either (it is *fully divergent*). c seems to be the perfect feature, because it separates all classes, but is fully conserved within the class. b is also good, because it can separate at least two subfamilies

$\{c_1, c_2\}$ vs $\{c_3, c_4\}$. e gets a negative weight, because it has the undesired property that it separates the proteins within a subfamily, but no between subfamilies. Note that b also gets the maximum weight, even though it can not separate all subfamilies. This property is desired in many cases where the number of subfamilies is larger than the twenty amino-acids, and one position can thus not possibly separate all subfamilies.

5.7.2 3D contacts

Multi-Relief can boost its performance by exploiting 3D structural information. The idea is that functional specificity in general is caused by multiple residues clustered close to each other. By adding the average weight of the neighbors that share surface with a residue to that residue, some performance gain can be achieved. This structural information can be obtained from a web server.

Multi-Relief has proved to be a state-of-the-art algorithm for finding functional specific residues in MSA's. It has a time-complexity similar to Iterated-Relief.

Chapter 6

New approach

Many sequence algorithms work on the assumption that within class conservation and inter-class divergence of residues is a strong lead for relevant features. All Relief-based algorithms discussed use this assumption to calculate the importance of a residue. However, the conservation of a feature within a subfamily may be more important than the difference between subfamilies or vice-versa. All Relief-based algorithms mentioned before do no make any distinction between these two properties; in other words conservation and divergence of residues is weighted equally.

It might be interesting to give a different weight to these two properties and find out what the result is on the predictive quality of the Relief-based algorithms. I will conduct experiments on multiple sequence datasets, using the latest state-of-the-art Relief-based algorithms.

6.1 Theory

Marchiori showed in her paper [22] that Relief can be decomposed into three different components. The NM of a sample x $W_{miss}(i)$, the maximum nearest hit $W_{hit,max}(i)$ and the minimum nearest hit $W_{hit,min}(i)$. Using weights for these three components one can emphasize the within class conservation or inter-class divergence of residues. These three components are defined as:

$$\begin{aligned} W_{miss}(i) &= \sum_{x \in X} (X(i) - NM_x(i)), \\ W_{hit,min}(i) &= min(W_{c1}(i), W_{c2}(i)), \\ W_{hit,max}(i) &= max(W_{c1}(i), W_{c2}(i)), \end{aligned} \tag{6.1}$$

where the classes $c1, c2$ are defined as:

$$W_{c1} = \sum_{x \in X, cl(x)=c1} (X(i) - NH_x(i)),$$

$$W_{c2} = \sum_{x \in X, cl(x)=c2} (X(i) - NH_x(i)), \tag{6.2}$$

Where X is a dataset, x is an alignment, i is a residue and $c1$ and $c2$ are the two subfamilies or classes used in Relief. The weight update part of the new Relief algorithm is thus defined as:

$$W_{new}(i) = \alpha_0 W_{miss}(i) - \alpha_1 W_{hit,min}(i) - \alpha_2 W_{hit,max}(i), \tag{6.3}$$

with $\alpha_0, \alpha_1, \alpha_2$ in $(0, 1]$. If all alphas are 1, the original weight update of Relief is obtained.

6.2 Heuristic AUC optimization

Marchiori has set the $\alpha_0, \alpha_1, \alpha_2$ values in her work, based on the assumption that the conservation of residues within a subfamily is more important than the divergence between subfamilies. Therefore, she has given the α_1 parameter (corresponding to the class with minimum intra-class distance for a position i and thus maximum conservation) twice as much weight as the other two parameters. It might be interesting to check which parameters optimize the performance of our prediction methods, by measuring the area under the ROC curve for each α_1 value.

I have done this by implementing the alpha parameters into the Relief-based algorithms discussed earlier. The used algorithms and their characteristics are summarized in table 6.1. I have conducted experiments on 18 well-studied sequence datasets, which are depicted in table 6.2. I have compared these results with the related work, discussed in chapter 4.

6.3 Research target

Relief basically consists of two important parts. The first part is finding the nearest neighbors NH_x and NM_x of an instance x from a given subfamily. The second is updating the weight for a residue position. By using equation 6.3, only the second part is changed [1]. Setting a higher value for α_1 means that you in fact 'penalize' cases where residues are not conserved. By doing this you can emphasize the importance of conservation by the weight you give to this penalty. This will result in lower (probably negative) weights. Since only the relative weight value is important, this has no negative effects.[2].

[1] This doesn't hold for Simba and I-Relief, they use a weighted distance and so if you affect the weights, you also affect the distance and the nearest neighbors that are chosen.

[2] This is actually not entirely true; I-Relief performs worse, which is explained in the discussion.

Now, using this information the goal is to find an answer to the question:

Is the within subfamily conservation of residues in a MSA equally important as the between subfamily divergence?

I have tried to find an answer to the following questions to answer this. They are implicitly answered in the remainder of this thesis.

- Is there a significant improvement in the computed AUC values for a higher value of α_1?
- If there is an improvement, does this hold for all used algorithms?
- If there is an improvement, does this hold for all datasets?
- How well does this method work, compared to other methods?

To say whether results are significant or not, I have defined a null-hypothesis and an alternative hypothesis:

- H_0: There is no significant improvement in the computed AUC values for different values of α_1.
- H_1: There is a significant improvement in the computed AUC values for different values of α_1.

Significance has been measured with a one-sided Student's TTest at a confidence level of 95% (See chapter 6.4.2).

6.4 Materials and methods

To find out the importance of within class conservation versus inter-class divergence, I have setup a large scale experiment. In this experiment I implemented 5 Relief-based algorithms (with 2 version of Multi-Relief and new Multi-Relief, resulting in a total of 7 algorithms) and tested this on 18 MSA datasets.

6.4.1 Materials

I have implemented the most successful Relief-based algorithms discussed in chapter 5 in a web program (for more information see appendix A) and conducted experiments on 18 datasets. 13 datasets are obtained from the authors of [7], the remaining 5 are obtained from the authors of [31]. Because *Simba* and *I-Relief* update the feature weight at each iteration, equation 6.3 can not be used for these algorithms. For *Relief-F*, equation 6.3 is more difficult to implement, because you'll have to sum over all classes. In datasets with many classes, the results might be skewed. Therefore for these algorithm, I 've simplified the formula to:

$$W_{new}(i) = \alpha_0 W_{miss}(i) - \alpha_1 W_{hit}(i) \tag{6.4}$$

Table 6.1 shows the used Relief-based algorithms and the way in which the parameter tuning is implemented. For *Multi-Relief* and *new Multi-Relief* both the hit_{min} / hit_{max} implementation from eq. 6.3 and the weight for only the nearest hit (eq. 6.4) have been implemented. This way, I can see if there are differences between these two implementations. All algorithms have equal time complexity except for *G-Flip* and *Iterative-Relief*. Results for *G-Flip* take too long to compute and it is considered inferior to *Simba*, so *G-Flip* is not further used for the experiments. The used datasets and their characteristics are depicted in table 6.2.

Nr	Algorithm	Abbr.	Time comp.Θ	Implementation
1.	Relief-F	RF	NMT	W_{hit}
2.	G-Flip	GF	N^2M^2T	not used
3.	Simba	Simba	NMT	W_{hit}
4.	Iterative-Relief	IR	N^2MT	W_{hit}
5.	Multi-Relief	MR	NM^2T	$W_{hit,min}$
6.	new Multi-Relief	nMR	NM^2T	$W_{hit,min}$
7.	Multi-Relief	MR*	NM^2T	W_{hit}
8.	new Multi-Relief	nMR*	NM^2T	W_{hit}

Table 6.1: Used Relief-based algorithms for the experiments. For *MR* and *nMR*, both the implementations using equation 6.3 and 6.4 are used. The other algorithms only use 6.4. In the column 'time complexity'; N is the alignment length, M the number of proteins and T the number of algorithm iterations.

6.4.2 Method

The algorithms from table 6.1 have been implemented in PHP. The datasets where loaded into a MySQL database. For the experiment I used 10 discrete values of $\alpha_1 \in [1..10]$. The other alpha values where kept at 1. After that, 10 experiments where run for every dataset-algorithm combination on every α_1 value. All algorithms where run at $t = 100$ iterations, except for I-Relief, which doesn't sample but runs over all proteins. For this algorithm the stopping criteria where set at $t = 5$ iterations (5 times a feature reweighing), or $\tau = 0.05$ (see section 5.6.2 for details).

For every dataset there is a list of known residues (from biological experiments). Using this list, I have computed AUC values for every α_1 value on every dataset-algorithm combination. Thus in total, 10 values of α_1 $\times$ 10 observations $\times$ 18 datasets $\times$ 7 algorithms = 12.600 AUC values have been computed.

Dataset	No of classes	Average class size	Min, max class size	No of sites	Site information	No known residues
cmb9	2	9.5	8, 11	196		7
cd00120	2	44.5	41, 48	1108		3
cd00264	2	15.5	12, 19	831		3
cd00333	2	13.5	10, 17	1118		12
cd00363	2	5.5	3, 8	591		6
cd00365	2	15	8, 22	1151		10
CN myc	2	17	7, 27	583		11
GPCR*	77	26.8	3, 189	214	Ligand	21
GST	11	9.7	9, 10	330	Protein	9
IDH/IMDH	4	17	8, 34	745		14
Laci*	15	3.6	2, 12	339	Ligand,DNA	28
MDH/LDH	2	22	9, 35	180		1
Nuc. cyc.	2	24.5	20, 29	231		2
Rab5/Rab6*	2	5.0	4, 6	162	Protein	28
Ras/Ral*	2	41.5	10, 73	217	Protein	12
Ricin	3	15.7	14, 19	135	Protein	21
Serine	3	32	6, 77	284		2
Smad*	2	16,5	16, 17	211	Protein	29

Table 6.2: Used sequence datasets. The datasets marked with * are from the authors of [31]. The others from the authors of [7]. Site information is only known for a small number of datasets.

For every algorithm and database, the significance over all 10 observations of every α_1 value has been compared to the case where $\alpha_1 = 1$ (the standard Relief). For the first sets I used both the Students TTest [29] and the Wilcoxon rank test [29] to compute p-values for the given observations to check if the mean over the observations significantly differed. It was soon clear that all observations follow a Gaussian distribution, so I continued with only the Students TTest. Using these p-values, I have drawn conclusions whether or not there is a significant performance increase.

If the average AUC over 10 observation of any $\alpha_1 > 1$ was higher than the AUC for $\alpha_1 = 1$ and $p < 0.05$, I've concluded that there was a significant performance increase. The goal is to find the α_1 that maximizes the AUC, and to keep the results compact only this value of α_1 is used in the result table 6.4. Trends are visualized in the plots in appendix B.

6.5 Results

The results are depicted in table 6.3 and table 6.4. The first contains the AUC values computed for the different datasets, the second contains the relative improvements (if there are any). For all datasets and all algorithms, the value of α_1 is given that maximizes the AUC, denoted as α_∞. The AUC

improvement relative to $\alpha_1 = 1$ is also given. If there is no significant improvement, this is denoted by a $*$ in that cell. The result tables for every algorithm-dataset combination can be found in the digital supplementary data that comes with this thesis. Because these are $7 \times 18 = 126$ tables, it didn't seem useful to put them all in the appendix. Plots that visualize the AUC fluctuations at different α_1 values can be found in appendix B.

Dataset	Known Res.	RF	Simba	IR	MR	nMR	MR*	nMr*	AVG
cbm9	7	0.522	0.340	**0.554**	0.477	0.477	0.537	0.475	0.483
cd00120	3	**0.970**	0.456	0.562	0.966	0.173	0.838	0.173	0.591
cd00264	3	0.469	0.244	**0.613**	0.542	0.165	0.296	0.165	0.356
cd00333	12	0.791	0.415	0.512	**0.883**	0.509	0.821	0.511	0.635
cd00363	6	0.493	0.505	**0.530**	0.478	0.507	0.507	0.504	0.503
cd00365	10	0.786	0.730	0.413	0.761	0.757	**0.800**	0.758	0.715
CN-myc	11	0.730	0.720	0.724	**0.759**	**0.759**	0.747	**0.759**	0.743
GPCR	21	0.808	0.803	-	0.771	0.877	0.699	**0.874**	0.805
GST	9	0.790	0.752	0.816	0.757	0.815	0.746	**0.816**	0.785
IDH/IMDH	14	0.735	0.721	0.774	0.735	**0.796**	0.751	0.792	0.758
Laci	28	0.760	0.799	0.798	0.788	0.794	0.455	**0.808**	0.743
MDH/LDH	1	**0.986**	0.795	0.961	0.968	0.963	0.965	0.960	0.943
Nucl.cycl.	2	0.785	0.548	**0.917**	0.777	0.863	0.856	0.865	0.802
Rab 5/6	28	0.685	0.692	0.732	0.680	0.737	0.726	**0.740**	0.713
Ras/Ral	12	0.575	0.653	**0.735**	0.546	0.612	0.570	0.613	0.615
Ricin	21	0.477	**0.509**	0.485	0.478	0.481	0.475	0.478	0.483
Serine	2	0.705	0.874	0.582	**0.917**	0.868	0.850	0.856	0.807
Smad	29	**0.965**	0.760	0.906	0.963	0.944	0.958	0.936	0.919
AVG		0.724	0.629	0.683	**0.736**	0.672	0.700	0.671	

Table 6.3: Computed AUC results for $\alpha_1 = 1$; best values are in bold. Average scores are in the last column and row. The relative performance increase is depicted in table 6.4.

Dataset	RF		Simba		IR		MR		nMR		MR*		nMR*		
	α_∞	AUC$_\%$	α_∞	AUC$_\%$	α_∞	$AUC_\%$	α_∞	$AUC_\%$	α_∞	$AUC_\%$	α_∞	$AUC_\%$	α_∞	$AUC_\%$	AVG
cbm9	1	0	6	+50.00	1	0	1	0	1	0	9	* +2.61	2	* +0.63	+7.14
cd00120	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0
cd00264	1	0	10	+86.89	1	0	1	0	1	0	1	0	1	0	+12.41
cd00333	1	0	1	0	2	+0.2	3	* +1.13	1	0	1	0	1	0	+0.03
cd00363	3	+7.30	1	0	2	+6.60	1	0	1	0	1	0	1	0	+1.99
cd00365	1	0	1	0	1	0	3	* +0.39	1	0	3	* +0.39	1	0	0
CN-myc	1	0	1	0	1	0	1	0	1	0	9	+1.07	1	0	+0.15
GPCR	1	0	1	0	-	-	1	0	1	0	1	0	1	0	0
GST	2	+2.15	1	0	4	+6.99	1	0	1	0	9	+5.23	1	0	+2.05
IDH/IMDH	2	+6.26	1	0	1	0	7	+2.86	1	0	2	+1.98	1	0	+1.59
Laci	2	+5.66	1	0	1	0	9	+2.54	1	0	6	* +3.52	1	0	+1.17
MDH/LDH	3	* +0.30	1	0	1	0	5	* +0.41	8	* +0.42	6	* +0.10	9	+0.73	+0.10
Nucl.cycl.	6	+16.56	9	+35.04	1	0	7	+18.66	10	+11.01	3	+5.26	10	+11.21	+13.96
Rab 5/6	4	+7.59	1	0	1	0	6	+1.03	1	0	1	0	1	0	+1.23
Ras/Ral	8	+16.17	2	+2.76	1	0	4	+2.93	4	+6.05	10	+5.61	5	+7.99	+5.93
Ricin	8	+7.34	3	* +2.55	1	0	10	+1.67	8	+6.44	6	+5.47	7	+8.16	+4.15
Serine	3	+3.55	1	0	1	0	1	0	7	+4.49	3	* +0.94	9	+6.07	+2.02
Smad	1	0	1	0	1	0	1	0	1	0	9	+0.21	1	0	+0.03
AVG		+4.03		+9.71		+0.77		+1.65		+1.56		+1.38		+1.90	

Table 6.4: Computed improvements for α_∞. The α_1 value that maximizes the AUC (α_∞) is given. Improvements of the AUC computed for α_∞ relative to α_1 are also provided. Results not significant for $p < 0.05$ are marked with a *. Average scores are in the last row and column. Results for IR on the GPCR dataset took too long to compute.

Chapter 7

Discussion

Tables 6.3 and 6.4 and the plots in appendix B contain a lot of data. I will discuss some of the most remarkable patterns here.

Explaining the results for $\alpha_1 = 1$

From the results in table 6.3, it can be concluded that Multi-Relief (MR) performs best over all datasets and Simba performs worst in the standard situation where $\alpha_1 = 1$. Note that MR and MR* are the same algorithms in the case that $\alpha_1 = 1$, the same holds for nMR and nMR*. This explains that there is almost no difference between the latter two. A slightly bigger difference between MR and MR* could be explained by the fact that in this case the averaging is only over the positive weights, which might vary a little more over 100 iterations. nMR* takes the average over all values which might explain the smaller differences. The number of known functional specificity residues seems to have no influence on the AUC values computed.

Conservation more important to shorter sequences?

As can be obtained from table 6.4 and the graphs in appendix B, most algorithms perform best when $\alpha_1 = 1$. On some datasets improvements have been achieved for some algorithms, and these are mostly the datasets with a relative short sequence length of about 135 to 330 amino-acids.

Three datasets stand out; *Nucleotidyl Cyclase*, *Ras/Ral* and *Smad* show an increasing predictive accuracy for every algorithm except IR (the reason for this is discussed below). For a non-biologist it is difficult to give an interpretation why this is so. However, the datasets with long sequences (over 1000 amino-acids) show no improvement for any algorithm. On the other hand, the datasets that benefit most from an emphasis on conservation of residues within subfamilies, are all relatively small and with limited sequence length. It is tempting to say that conservation is more important for shorter sequences. The length of a protein has probably a relation to its

function and therefore there might be a good biological explanation for this remarkable result.

IR performance deteriorates for higher α_1 values

Strikingly, IR shows almost no improvement for any $\alpha_1 > 1$. A first thought might be that this has something to do with the feature reweighing. Therefore it affects the nearest neighbor computation at every new iteration, which could explain this result. However, Simba also incorporates this method of feature reweighing and shows some remarkable improvements on some datasets. The reason that IR doesn't improve for most α_1 values bigger than 1 is probably because it optimizes the margin and doesn't allow a negative margin. From equation 5.17, one can easily see that all negative weights are set to 0. And from equation 6.4, one can easily see that increasing the value of α_1, increases the number of negative weights even if only a small number of residues are not conserved. In this case, most residues -relevant or not- get weight 0, which results in lower predictive power and corresponding lower AUC values. This pattern is nicely illustrated by the graphs in appendix B.

Simba has some remarkable performance increases

Simba shows some remarkable improvements of 35% to even 86%. On other datasets, improvements are only small or negative. This might have something to do with the reweighing discussed above, although it is strange that it only occurs in three cases. If you look at table 6.3, you can conclude that it started with an arrear for $\alpha_1 = 1$. In all three cases, it performed much worse than the other algorithms. Although it improves much for higher α_1 values, it still doesn't outperform the other algorithms.

Deteriorating predictive accuracy

A result that can easily be obtained from table 6.4 is that there is no clear α_1 value that maximizes the AUC. Increasing α_1 from 1 to 2 might result in an improvement, but increasing α_1 to higher values often deteriorates the predictive accuracy. Thus there is no boundless increase in predictive accuracy by just increasing the value of α_1. This seems reasonable because at high α_1 values conservation is so much emphasized that divergence can almost be neglected and thus we sort of lose this information. The graphs in appendix B clearly show this trend. It might be interesting to turn the method around by giving more weight to the inter-class divergence and compute the corresponding AUC results. Unfortunately I had no time to perform these experiments, but it would be interesting for future research.

Results compared to other methods

The authors of Multi-Harmony [5] have included an extensive table with all computed AUC results for all datasets mentioned in the previous chapter. However, the AUC results they computed are hard to compare because they seem pretty low or are normalized somehow. The AUC results they computed with the Multi-Relief method also significantly differs from the results in [31].

My results from table 6.3 and the results in [31] show more agreement. The slightly better scores in the latter can be explained by the number of iterations used (100 in this thesis versus 1000 in [31]).

Chapter 8

Conclusions

Relief-based algorithms can weight features according to the conservation of residues within a subfamily and the divergence between subfamilies in a given MSA. The assumption that conservation is more important than divergence is only partially confirmed by the experiments. Remarkable is that especially proteins of smaller length seem to benefit from more emphasis (expressed as a higher value of α_1) on within subfamily conservation of residues. In general, this trend is visible on all implemented algorithms, except for Iterative-Relief. The reason is that this algorithm computes a margin and therefore doesn't allow negative weights, which strongly affects the feature weighting method.

Another interesting conclusion is that there is no clear maximum for the α_1 value I've tried to optimize. Increasing α_1 without bound is useless, because in general the computed AUC results drop after a certain optimum value, strongly dependent on the dataset and algorithm used. A sound explanation is that by emphasizing conservation too much the inter-class divergence gets oppressed. This means that for high α_1 values the divergence is almost neglected, which is lost information that results in lower predictive accuracy. This makes it difficult to incorporate this method into new algorithms, because the optimum settings are not known a priori.

Unfortunately, only conservation has been emphasized. It might be interesting to conduct similar experiments where inter-class divergence is weighted more important than within-class conservation. It would be really interesting to see if proteins with larger length benefit in this case, which would support my research results.

New directions

Proteins are very complex molecules, with a complex structure. Many properties are even unknown today. So modeling these molecules with a simple string representation, which a MSA in fact is seems somewhat limited. Improving algorithms based on MSA's only will inevitably reach its ceiling

soon. In my opinion methods for predicting residue specificity should include more data if possible. Phylogenetic trees, chemical data, amino-acid interaction matrices and 3D information are other data sources that could be valuable to incorporate in new models. In fact methods like SPEER [7] incorporate multiple methods and data sources to compute specificity determining residues. By combining Relief-based methods for MSA's with new research methods, we could combine the best of both worlds and obtain better results. Much is to be discovered...

Appendix A

Webserver implementation

To perform the experiments in this thesis, I have created a web application in PHP using Zend Framwork [1]. I have used PHP because it is fast and the framework assists me in creating a nice class-hierarchy for my algorithms, that can easily be extended in the future if desired. I have chosen a web application to enable other researchers to use the program. They can easily upload their own data and conduct the experiments they want. The application has two main functions:

- Select features on a given dataset. The result is a list of features and their computed weights;
- Compute the AUC for a given dataset on different values of α_1.

A.1 Feature selection on a given dataset

The first function is a basic implementation of the feature selection algorithms. You can either select a database or upload your own sequence in FASTA-format, set some parameters and run the application to obtain the computed feature weights. Figure A.1 shows a screenshot of this function.

A.1.1 Input

The input for this functionality is a given MSA in FASTA-format, a division into the subfamilies used and optionally a list of known functionality specific residues (true positives). The 18 datasets I used for my experiments are preloaded, including their subfamily division and true positives. Users can also just select one of these datasets from a dropdown box. In either case the parameters to select are:

- **Algorithm:** the Relief-based algorithm you'd like to use;

[1] http://framework.zend.com/

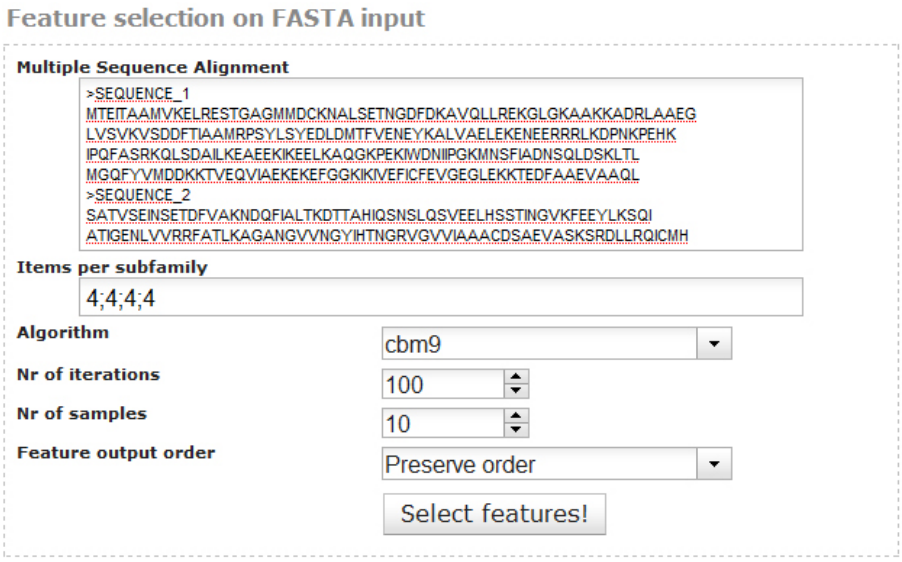


Figure A.1: Input page for a MSA in FASTA format. Users can upload their own MSA's here and choose the parameters to use for discovering the functional specificity residues.

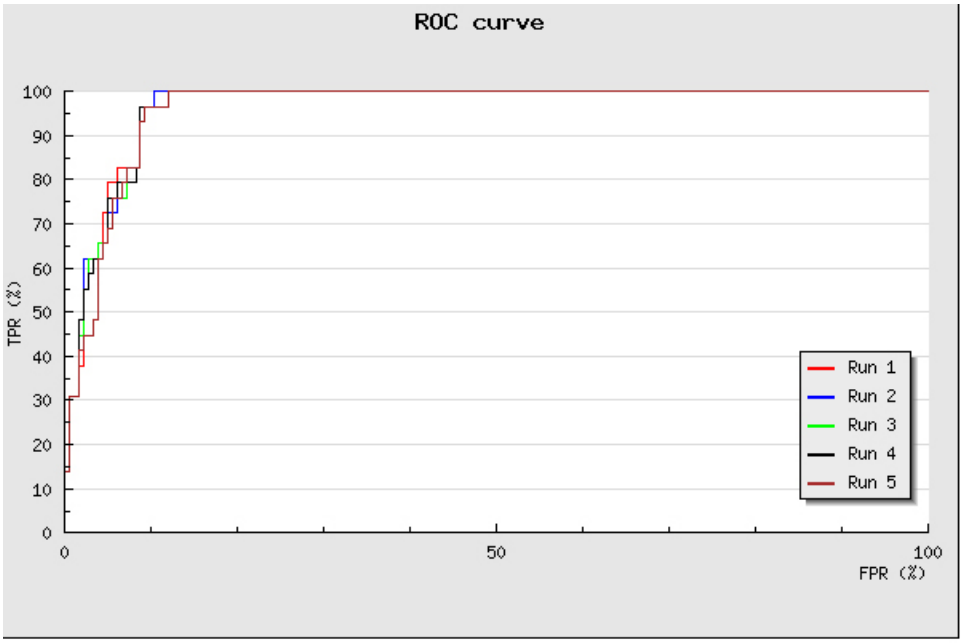
- **Nr iterations:** the number of iterations for the selected algorithm;
- **Nr samples:** the number of samples to use, if the algorithm is based on (sub-)sampling, otherwise this parameter can not be selected;
- **Feature output order:** whether to preserve the input order, or order features by descending weight.

A.2 Compute AUC on given α_1 value

This function doesn't return a weighted feature list, but plots a ROC-curve for different α_1 values. The ROC-curves for $\alpha_1 = 1$ to $\alpha_1 = x$ are plotted, where x can be any discrete value between 1 and 10. This functionality has the same input and parameters as the one in the previous section, with one extra:

- **Alpha-1 max:** The maximum value of alpha to use, AUC's are computed for every α_1 from 1 to *Alpha-1 max*. The maximum is limited to 10, which is the value used for my experiments. This is done for both efficiency reasons and because experiments pointed out that higher values show no improvements.

The output is a plotted ROC-curve and a table with the computed AUC values (see A.2 for an example) and there corresponding values of α_1.



Run	Alpha 0	Alpha 1	Alpha 2	AP	AUC
1	0.20	0.20	0.50	0.786	0.967
2	0.45	0.50	0.50	0.778	0.965
3	0.45	0.55	0.50	0.779	0.965
4	0.45	0.50	0.50	0.780	0.965
5	0.50	0.50	0.50	0.762	0.962

Figure A.2: Example of a plotted ROC-curve. This ROC-curve was potted using Relief-F on the SMAD-receptors dataset. In this example, unlike in the experiments conducted the other α values are not kept at 1.

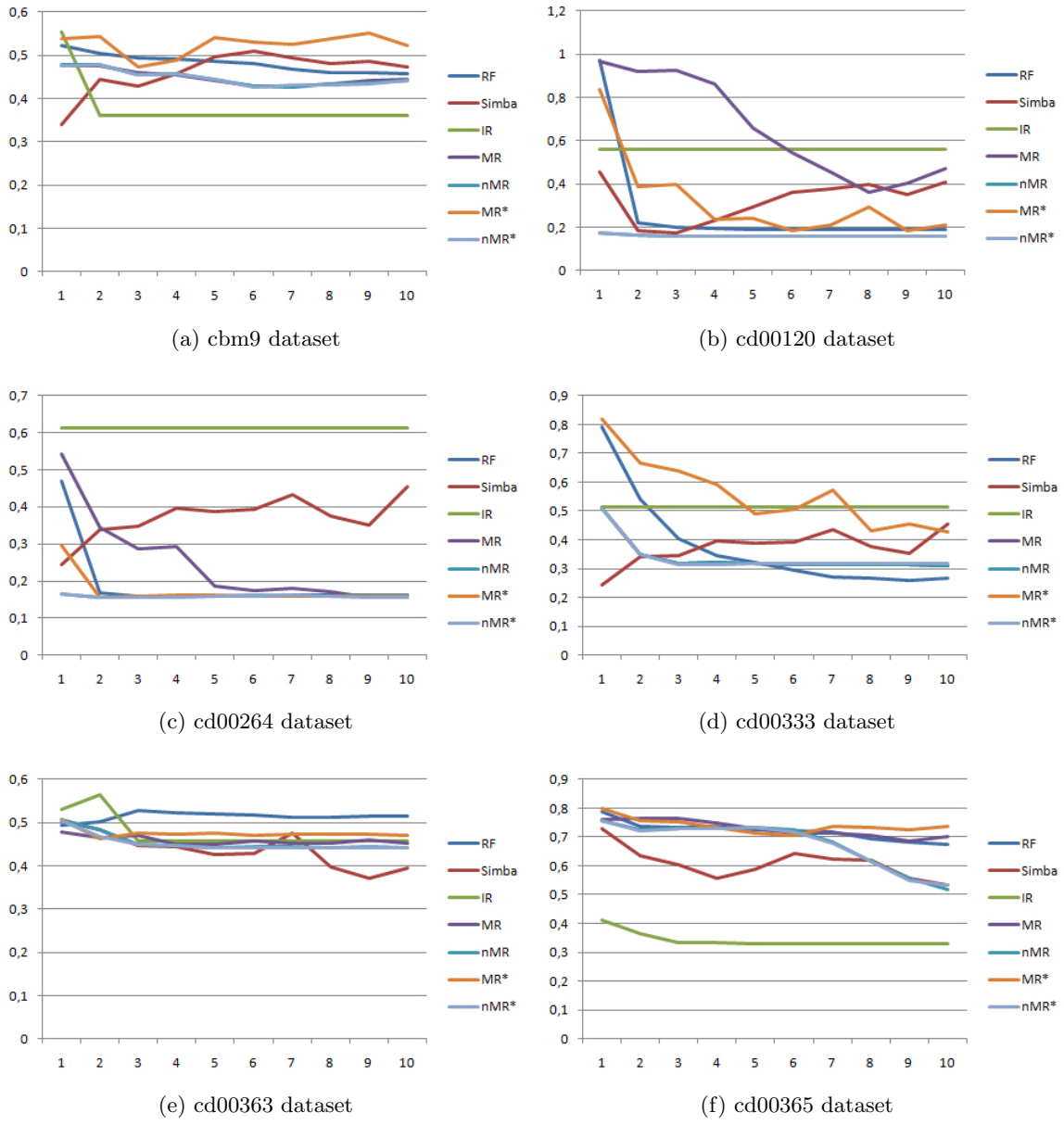
A.2.1 Additional material

The web application includes a short manual and tutorial to guide beginning users. A documented API and UML class schema are also provided, to enable users to change or extend this method. At the time of writing this thesis, the web application is used local and therefore a URL can not yet be provided.

Appendix B

AUC plots for different α_1 values

In this appendix, the area under the ROC-curve (AUC) values for $\alpha_1 \in [1..10]$ are plotted for each algorithm used on all the 18 datasets. On the horizontal axis the values of α_1 values are plotted. The corresponding AUC value is plotted on the vertical axis. The name of the dataset used is depicted under each plot.



(a) cbm9 dataset

(b) cd00120 dataset

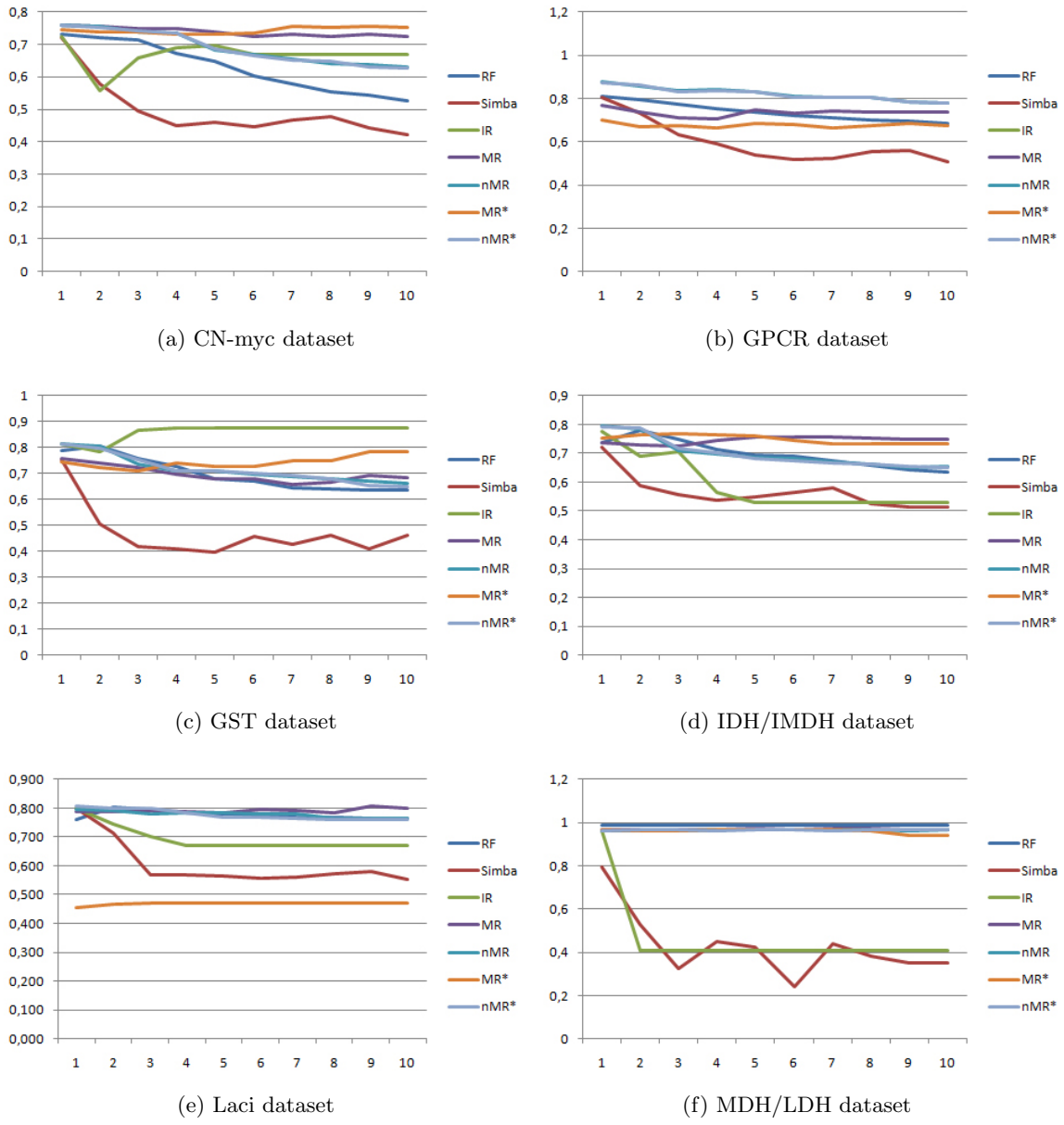
(c) cd00264 dataset

(d) cd00333 dataset

(e) cd00363 dataset

(f) cd00365 dataset

Figure B.1: AUC values plotted for different α_1 values. The α_1 values are plotted on the x-axis, the corresponding AUC values on the y-axis.



(a) CN-myc dataset

(b) GPCR dataset

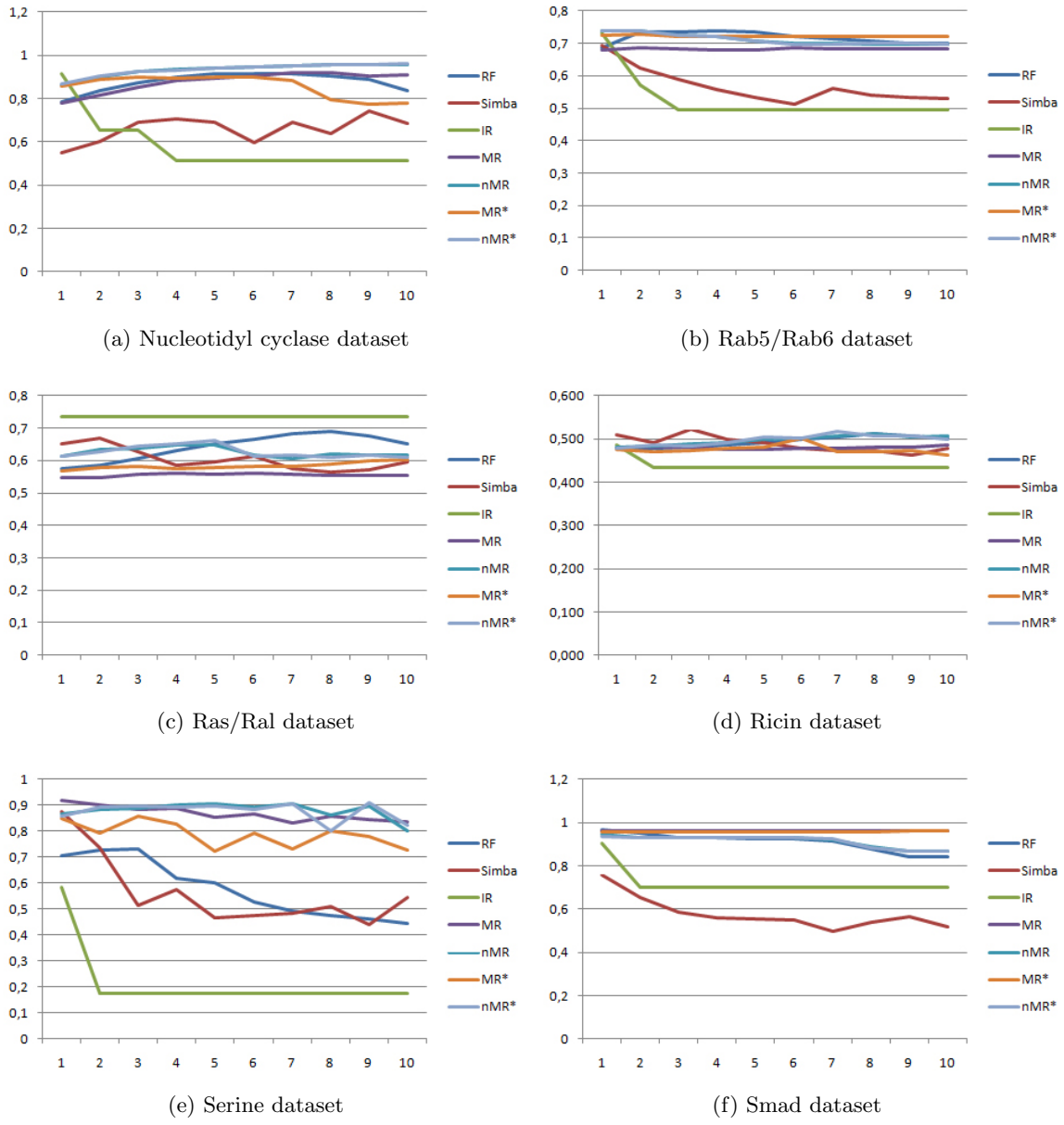
(c) GST dataset

(d) IDH/IMDH dataset

(e) Laci dataset

(f) MDH/LDH dataset

Figure B.2: AUC values plotted for different α_1 values. The α_1 values are plotted on the x-axis, the corresponding AUC values on the y-axis.



(a) Nucleotidyl cyclase dataset

(b) Rab5/Rab6 dataset

(c) Ras/Ral dataset

(d) Ricin dataset

(e) Serine dataset

(f) Smad dataset

Figure B.3: AUC values plotted for different α_1 values. The α_1 values are plotted on the x-axis, the corresponding AUC values on the y-axis.

Bibliography

[1] Achuthsankar, S. Computational biology and bioinformatics - a gentle overview. *Communications of Computer Society of India* (2007).

[2] Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. Basic local alignment search tool. *Journal of Molecular Biology 215*, 3 (1990), 403–410.

[3] Baldi, P., and Brunak, S. *Bioinformatics: The Machine Learning Approach*. The MIT Press, 1998.

[4] Blum, A. L., and Langley, P. Selection of relevant features and examples in machine learning. *Artificial Intelligence 97*, 1-2 (1997), 245–271.

[5] Brandt, B. W., Feenstra, K. A., and Heringa, J. Multi-Harmony: detecting functional specificity from sequence alignment. *Nucleic Acids Research 38*, suppl2 (2010), W35–40.

[6] Breiman, L., Friedman, J., Olshen, R., and Stone, C. *Classification and Regression Trees*. Wadsworth and Brooks, 1984.

[7] Chakrabarti, S., Bryant, S. H., and Panchenko, A. R. Functional specificity lies within the properties and evolutionary changes of amino acids. *Journal of Molecular Biology 373*, 3 (2007), 801–810.

[8] Cheng, B. Y. M., Carbonell, J. G., and Klein-Seetharaman, J. Protein classification based on text document classification techniques. *Proteins : structure, function and genetics 58*, 4 (2005), 955–970.

[9] Gilad-Bachrach, R., Navot, A., and Tishby, N. Margin based feature selection - theory and algorithms. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning* (2004), ACM, p. 43.

[10] Guyon, I., and Elisseeff, A. An introduction to variable and feature selection. *The Journal of Machine Learning Research 3* (2003), 1157–1182.

[11] Guyon, I., Gunn, S., Nikravesh, M., and Zadeh, L., Eds. *Feature Extraction, Foundations and Applications*. Springer, 2006.

[12] Hannenhalli, S. S., and Russell, R. B. Analysis and prediction of functional sub-types from protein sequence alignments. *Journal of Molecular Biology 303*, 1 (2000), 61–76.

[13] John, G. H., Kohavi, R., and Pflegger, K. Irrelevant features and the subset selection problem. In *Machine learning: Proceedings of the Eleventh International Conference* (1994), Morgan Kaufmann, pp. 121–129.

[14] Kalinina, O. V., Mironov, A. A., Gelfand, M. S., and Rakhmaninova, A. B. Automated selection of positions determining functional specificity of proteins by comparative analysis of orthologous groups in protein families. *Protein Science 13*, 2 (2004), 443–456.

[15] Kamleitner, B. Machine learning feature weighting algorithms for discovering functionally specific residues, master thesis, 2007.

[16] Kira, K., and Rendell, L. A. The feature selection problem: Traditional methods and a new algorithm. In *Proceedings of the AAAI-92* (1992), AAAI Press, pp. 129–134.

[17] Kira, K., and Rendell, L. A. A practical approach to feature selection. In *The 9th International Conference on Machine Learning* (1992), Morgan Kaufmann, pp. 249–256.

[18] Kohavi, R., and John, G. H. Wrappers for feature subset selection. *Artificial Intelligence 97*, 1-2 (1997), 273–324.

[19] Kononenko, I. Estimating attributes: Analysis and extensions of relief. In *European Conference on Machine Learning* (1994), Springer Verlag, pp. 171–182.

[20] Liu, H., and Motoda, H. *Feature selection for knowledge discovery and data mining*. Kluwer Academic Publishers, 1998.

[21] Liu, H., and Motoda, H. *Computational Methods of Feature Selection*. Chapman & Hall/CRC, 2008.

[22] Marchiori, E. *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*. Springer, 2010, ch. Improving Multi-Relief for Detecting Specificity Residues from Multiple Sequence Alignments, pp. 158–169.

[23] Pazos, F., Rausell, A., and Valencia, A. Phylogeny-independent detection of functional residues. *Bioinformatics 22*, 12 (2006), 1440–1448.

[24] Peng, H., Long, F., and Ding, C. Feature selection based on mutual information: Criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence 27*, 8 (2005), 1226–1238.

[25] Pirovano, W., Feenstra, K. A., and Heringa, J. Sequence comparison by sequence harmony identifies subtype-specific functional sites. *Nucleic Acids Research 00*, 00 (2006), 1–9.

[26] Quinlan, J. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers, 1993.

[27] Saeys, Y., Inza, I., and Larran, P. A review of feature selection techniques in bioinformatics. *Bioinformatics 23*, 19 (2007), 2507–2517.

[28] Sun, Y., and Li, J. Iterative relief for feature weighting. In *ICML '06: Proceedings of the 23rd international conference on Machine learning* (2006), ACM, pp. 913–920.

[29] Wonnacott, T. H., and Wonnacot, R. J. *Introductory statistics*. John Wiley and Sons, 1990.

[30] Yang, Y., and Pedersen, J. A comparative study on feature selection in text categorization. *In Proceedings of ICML-97, 14th International Conference on Machine Learning* (1997), 412–420.

[31] Ye, K., Anton Feenstra, K., Heringa, J., IJzerman, A. P., and Marchiori, E. Multi-relief. *Bioinformatics 24*, 1 (2008), 18–25.

[32] Yu, L., and Liu, H. Efficient feature selection via analysis of relevance and redundancy. *Journal of Machine Learning Research 5* (2004), 1205–1224.

Glossary

alignment An alignment is a sort of ordering where residues or amino-acids between proteins are as much conserved as possible. 5

conserved residues Residues that has the same value at a given position for all proteins in a subfamily of a MSA. 5, 6

Hamming distance The Hamming distance in the context of this thesis is measured between amino-acids. When two amino-acids have equal value, the corresponding Hamming distance is 0. If the values are not equal, the Hamming distance is 1. 25

MSA A multiple sequence alignment (MSA) is a dataset where all protein sequences are aligned. 2, 5

nearest hit The nearest hit for a given protein x is the nearest neighbor from the same subfamily. 25

nearest miss The nearest miss for a given protein x is the nearest neighbor from the opposite subfamily. 25

nearest neighbor The nearest neighbor for given protein x is the protein with the smallest distance to x. The distance in this case is the Hamming distance summed over all amino-acids. 25

residue A residue is one amino-acid in a sequence. It is thus a synonym for both an amino-acid and a feature in the context of this thesis. 4

sequence A sequence is an ordered string of all the amino-acids in a protein. 4