Fachhochschule Weihenstephan
Fachbereich Biotechnologie

Diplomarbeit

Sequence-based feature prediction on proteins

Michael Kleen

2006

Erklärung zur Urheberschaft Gemäß § 31 Abs. 7 der Rahmenprüfungsordnung für die Fachhochschulen (RaPO): Ich erkläre hiermit, dass die vorliegende Arbeit von mir selbst und ohne fremde Hilfe verfasst und noch nicht anderweitig für Prüfungszwecke vorgelegt wurde. Es wurden keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt. Wörtliche und sinngemäße Zitate sind als solche gekennzeichnet.
Cambridge, den 12.10.2006

Michael Kleen

First of all, I would like to say thanks to Ernst Kretschmann for the brilliant supervision of this thesis, it was a real pleasure to work in his group. He also gave me a lot of private assistance for which I am very grateful. Also thanks to Daniela Wieser and Marcus Ennis for all the fruitful discussions and the proofreading of the thesis. Finally I would like to thank Prof. Dr. Bernhard Haubold for the feedback on an earlier version of this thesis.

Contents

1 Summary **8**

2 Introduction **9**

2.1 Motivation for this work . . . 9

3 Methods **15**

3.1 Decision trees . . . 15

3.1.1 Divide and conquer . . . 15

3.1.2 Choosing the best attribute . . . 20

3.1.3 Confidence score . . . 23

3.2 Finite state machines . . . 24

3.2.1 Acceptor . . . 25

3.2.2 Transducer . . . 26

3.2.3 Weighted acceptor . . . 27

3.2.4 Weighted transducer . . . 28

3.2.5 Operations . . . 29

3.2.6 Building a predictive model . . . 33

3.3 Summary . . . 38

4 Implementation **39**

4.1 Algorithm module . . . 40

4.1.1 Decision trees . . . 40

4.1.2 Weighted finite state machines . . . 42

4.2 Workflow module . . . 43

4.3 Backend module . . . 49

4.4 Web module . . . 50

5 Results **54**

5.1 Dataset . . . 54

5.2 Cross-validation . . . 55

5.3 Four-fold cross-validation in Swiss-Prot . . . 56

5.4 Ten-fold cross-validation in the enolase family . . . 60

5.5 Current state of the system . . . 62

6 Discussion **63**

6.1 How reliable are the results ? . . . 63

6.2 Comparison with structure based methods . . . 64

6.3 Possible Improvements for the algorithm . . . 64

6.4 Application to TrEMBL proteins . . . 65

7 Conclusion **66**

8 Appendix **67**

8.1 Databases involved . . . 67

8.1.1 Swiss-Prot . . . 67

8.1.2 TrEMBL . . . 67

8.1.3 InterPro . . . 67

8.1.4 Prints . . . 68

8.1.5 Prosite . . . 68

8.1.6 Pfam . . . 69

8.2 Swiss-Prot datatypes . . . 69

8.2.1 ID . . . 69

8.2.2 DE . . . 69

8.2.3 OS . . . 69

8.2.4 RN, RP, RC, RX, RA, RT, RL . . . 70

8.2.5 CC . . . 70

8.2.6 DR . . . 70

8.2.7 KW . . . 70

8.2.8 FT . . . 71

8.2.9 SQ . . . 71

9 Bibliography **72**

1 Summary

Molecular functions of proteins depend on the properties of their primary sequences, in particular on some site-specific functional residues. Thus, site-specific annotations from the UniProt Knowledge base (UniProtKB), (Wu et al., 2006) constitute a highly valuable resource in a standardized format and are therefore important and central annotations. Unfortunately they are unavailable for most proteins in the UniProtKB.

There are several well proven methods to detect molecular functions on proteins but these methods do not have much space left for further optimization and can not distinguish whether the execution of a predictive model is useful or not. They also do not provide functional description in a standardized way such as in the UniProtKB. For this reason a combination of the well known C 4.5 decision tree algorithm and a weighted finite state machine approach was used taking taxonomic and sequence specific data to build the predictive models. These models produce position-specific functional site annotations in UniProtKB format.

The method was tested in a cross-validation against all active site annotations in UniProtKB/Swiss-Prot. A recall rate of 72.47% was obtained at a precision of 96.9%. The adaptability of the algorithm to other functional site annotations was evaluated for substrate binding regions and metal binding annotation in the enolase protein family. Finally, an automated annotation pipeline was implemented which can be used to run the algorithm whenever updated or new training data is available.

2 Introduction

2.1 Motivation for this work

Proteins are essential in every life form. They control most of the molecular processes in a living cell and they are involved in nearly every function. Signal transduction and metabolism are common examples where proteins play are an essential role, therefore the most interesting information about a protein is its biological function. So what is the biological function of a protein and why is it so important? Every protein performs a specific function within an organism and this has consequences from the subcellular to the whole-organism level. Depending on these functions, proteins can be grouped into specific classes. Also, the function of a protein is highly dependent on its properties. Proteins have properties on different levels and the most important one of these is structure.

There are four different aspects of a protein structure. The primary structure is the amino acid sequence. The secondary structure describes the general three-dimensional form of local segments of biopolymers such as proteins and nucleic acids. The tertiary structure is the overall shape of a single protein molecule and the quaternary structure is the shape that results from the union of more than one protein molecule which function as part of the larger assembly or protein complex. In addition to these levels of structure, proteins may shift between several similar structures in performing their biological function. They share conserved regions which are called protein domains and which indicate a specific function. A conserved region is a sub-sequence that shares high similarity between a number of proteins which have similar functions. Various molecules and ions are able to bind to specific sites on proteins. These sites are called binding sites and they are indicators for the function of a protein. If, for an example, there is one zinc atom bound to two cysteine and two histidine amino acid residues then this is thought to indicate the involvement of the protein in a DNA/RNA interaction.

Therefore, a raw protein sequence without any further information about the behaviour of that protein is not very helpful to a biologist. However, since there is an exponential increase in the number of proteins being identified from large-scale sequence genome projects it is becoming far more difficult to characterize all these proteins by functional assays. For this reason, protein functional predictions have arisen in computational biology. In this field, functional prediction methods are developed to predict the function of a protein. There is a need for developing these new methods, because there is a widening gap between known protein sequences and their functions.

Protein sequences are represented as character strings, so many of these methods try to find similarity patterns based on the amino acid sequence. Mostly string-matching methods like regular expressions (Hopcroft and Ullman, 1979), hidden Markov models (Rabiner and Juang, 1986), and dynamic programming are used for this approach. Proteins can be classified using these methods of detecting conserved regions such as protein domains, protein families, and binding sites. These conserved regions on the protein sequence are called signature hits. Each method normally describes a specific amino acid sequence pattern where a biological meaning is assigned by a human curator. For example the signature hit PF01361 detected by the hidden Markov models of the Pfam database (Bateman et al., 2004) classifies a protein to the tautomerase enzymes. This indicates that the protein is an enzyme which catalyses the ketonization of 2-hydroxymuconate to 2-oxo-3-hexenedioate.

Unfortunately there are some problems with these methods. One problem is that every method has to be optimized to a specific sequence pattern. Normally one method finds only one specific sequence pattern. This means a conglomeration of predictive models needs to be applied to archieve a better understanding of the protein sequence. The Pfam database contains currently 8295 models for the detection of protein families on protein sequences. Each protein family is represented by curated multiple-sequence alignments and a corresponding hidden Markov model. Applying all models from Pfam to a single protein sequence of 300 amino acids takes about a minute on a single machine.

There are various methods available which have different strengths and weaknesses depending on their underlying analysis methods. A popular system that combines several predictive methods is InterProScan (Quevillon et al., 2005). This system uses the predictive models from several signature-hit databases such as Pfam (Bateman et al., 2004), Prosite (Hulo et al., 2006), Prints (Attwood et al., 2003), and Smart (Letunic et al., 2006) to analyze raw protein sequences. InterProScan applies 19484 models and it takes over 4 minutes for a protein of 300 amino acids. InterProScan is regularly applied to the over 3 million protein sequences of the UniProt Knowledge base (UniProtKB), (Wu et al., 2006). The runtime behaviour of InterProScan has a linear-complexity with an increase in the sequence length, but still the application of the system to protein sequences is a very time-consuming task. A closer look at the underlying procedures for the signature hits shows that the the problem is not the efficency of the methods, but the exponential increase in protein sequences from high-throughput sequencing projects. The creation of the models can be difficult but the application to the sequence is easy and very fast. The most popular method that is mainly used for detecting patterns in protein sequences is that of hidden Markov models. Hidden Markov models have their origin in speech recognition and have been optimized over decades. Once a hidden Markov model is created, regardless of its complexity, standard algorithms can be used for aligning and scoring sequences. These algorithms, e.g. Forward, have a worst-case complexity of O(NM2) in time and O(NM) in space where N is the sequence length of the protein and M the number of states of the hidden Markov model (Eddy, 1998). The problem is that much calculation time is wasted by applying specific methods on protein sequences where it is not determined if these are useful or not. As an example it is not useful to apply a method which tries to find an active site on a protein if the protein is not an enzyme.

An alternative approach to gain useful information about protein functions is the use of data-mining methods in combination with protein databases. One of the most popular database is the UniProtKB. The UniProtKB contains over 3 million proteins described in entries using a controlled vocabulary and consists of two parts: The Swiss-Prot section contains 230000 protein entries containing fully manually annotated records, the annotation resulting from literature information extraction carried out by human experts supported by computer-based analysis tools. The TrEMBL section contains two million protein entries and is by far the largest part of the database. The entries are computer-annotated and derived from the translation of all the coding sequences in the nucleotide sequence databases. Figure 2.1 shows parts of a UniProtKB entry from the Swiss-Prot section:

```
IID   XYL1_CANPA     STANDARD;      PRT;   324 AA.
AC   Q6Y0Z3;
DT   11-OCT-2004, integrated into UniProtKB/Swiss-Prot.
DE   NADH-dependent D-xylose reductase (EC 1.1.1.175) (XR).
GN   Name=XYL1;
OS   Candida parapsilosis (Yeast).
OC   Eukaryota; Fungi; Ascomycota; Saccharomycotina; Saccharomycetes;
OC   Saccharomycetales; mitosporic Saccharomycetales; Candida.
      .
      .
CC   -!- FUNCTION: Reduces D-xylose into xylitol. Preferentially utilizes
CC       NADH as a cosubstrate.
CC   -!- CATALYTIC ACTIVITY: D-xylose + NAD(+) = D-xylonolactone + NADH.
CC   -!- PATHWAY: D-xylose degradation.
CC   -!- SIMILARITY: Belongs to the aldo/keto reductase family.
     .
     .
DR   InterPro; IPR001395; Aldo/ket_red.
DR   Pfam; PF00248; Aldo_ket_red; 1.
DR   PRINTS; PR00069; ALDKETRDTASE.
DR   ProDom; PD000288; Aldo/ket_red; 1.
DR   PROSITE; PS00798; ALDOKETO_REDUCTASE_1; 1.
DR   PROSITE; PS00063; ALDOKETO_REDUCTASE_3; 1.
KW   Carbohydrate metabolism; Direct protein sequencing; NAD;
KW   Oxidoreductase; Xylose metabolism.
FT   CHAIN         1    324       NADH-dependent D-xylose reductase.
FT                                /FTId=PRO_0000124659.
FT   NP_BIND     220    286       NAD (By similarity).
FT   ACT_SITE     54     54       Proton donor (By similarity).
FT   BINDING     116    116       Substrate (By similarity).
FT   SITE         83     83       Lowers pKa of active site Tyr (By
FT                                similarity).
SQ   SEQUENCE   324 AA;  36629 MW;  C64951D131707E19 CRC64;
      MSTATASPAV KLNSGYEIPL VGFGCWKLTN DVASDQIYRA IKSGYRLFDG AEDYANEQEV
      LIRGCTIKPV ALQIEHHPYL TQPKLVEYVQ LHDIQITGYS SFGPQSFLEM DLKRALDTPV
```

Figure 2.1: The entry Q6Y0Z3 describes a NADH-dependent D-xylose reductase.

Functional descriptions of proteins are supplied in this entry as so-called *protein features* and are derived through a literature curation process, which is supported by expert-evaluated computational analysis. In Knowledge base entries, these are combined with other annotations, such as protein names, and EC numbers in the description line (DE), comments, shown in the description line (DE), the keyword line (KW), and the comments line (CC). Manual curation provides consistent annotation for all members of the same protein family. For instance, *D-xylose 1-dehydrogenases* are enzymes, which catalyze the reaction *D-xylose + NAD+ = D-xylonolactone + NADH + H+*, a fact that is given as a textual comment for all Swiss-Prot proteins belonging to this family. Enzyme related reactions are standardized by the *Nomenclature Committee of the International Union of Biochemistry and Molecular Biology (NC-IUBMB)* and a reference to the corresponding *EC 1.1.1.175* is given in the descriptions of all these proteins. There are also references to the pathway in which they are involved, including which step they catalyze (*D-xylose degradation*), and to which protein family they are associated (*belongs to the aldo/keto reductase family*). Keywords characterize the function in a more general way. For the *NADH-dependent D-xylose reductase*, the keywords *Carbohydrate metabolism*, *NAD*, and *Oxidoreductase* are used as subject references for many of these proteins. For location-specific annotation, a selection of features (FT) is used: *BINDING* for individual residues involved in non-covalent interactions with non-protein molecules, *NP_BIND* for regions of residues defining the nucleotide binding site of NAD, and *ACT_SITE* for the active site residues directly involved in catalysis. The entry also provides references to detected signature hits from the various databases. Unfortunately the much larger TrEMBL section of the UniProtKB provides for most of the proteins only a description of the host and the taxonomy along with signature hits and references to databases.

The Spearmint system (Kretschmann et al., 2001) uses the annotations described above to generate rules for protein annotations. The system learns on the Swiss-Prot part of the UniProtKB and uses signature hits and the taxonomy of the host organisms of the proteins to create decision trees (Quinlan, 1988). The rules from these decision trees are used to detect annotations, such as protein names, EC numbers, keywords or comments in the much larger TrEMBL part of the UniProtKB. Spearmint can detect these annotations on an uncharacterized protein from the TrEMBL section. Using this procedure regulary on the UniProtKB, the usefulness of the TrEMBL section of the UniProtKB can be increased considerably. The method has several advantages. First the method is highly efficient, a complete run of the system on the 3 million proteins of the UniprotKB base is taking about 6 hours on a 40 node cluster. Thus the performance for a single protein is less than one second. Secondly, the system contains low error rates, less than about 5 % with a coverage of 78 %. The benefits of this method is that the predicted annotations are more useful than signature hits. Since there is a textual description of the pattern presented in a standardized format while every signature hits database provides an format a description of the detected pattern in its own format. Unfortunately , the method works only at an annotation level rather than on a structure level, so only the presence of a function and not the exact position at the primary structure level can be predicted.

Therefore the problem is to find a method which detects the function of a protein at its primary structure level and also is efficient enough to work on large-scale data sets. String-matching methods like hidden Markov models are already very well optimized so there is not much space for optimizing these methods further. Binary classifying systems such as decision trees or neural networks are efficient methods which in several cases have been successfully applied for predicting functional behaviour. However, such methods cannot be used for detecting the position of a functional annotation. Hence, in this work the following approach was taken to generate a position-specific predictive model for detecting functional sites on protein sequences.

To get the benefits of both methods a two-stage approach is suggested, in order to achieve high efficiency by avoiding any time-consuming attempts at finding a particular molecular function in a protein that does not contain such a function. The well known decision tree algorithm from the automated annotation can be used to check for the presence of a functional site based on the core data from the UniprotKB. If this part of the algorithm is positive, a predictive model based on the primary structure of the of the positive instances from the training set is trained to detect the position of a protein feature. This approach has several advantages: Only proteins which might contain a specific molecular function are analyzed at the sequence level. It also reduces the complexity required for a position-finding model, if it can be assumed that only sequences actually containing the functional site are ever used with the model.

3 Methods

For the suggested two-stage approach, where one algorithm is used to predict the presence of a molecular function while the another is used for detecting the position at the sequence level, two completely different methods need to be combined into one aggregated system. The following chapter gives an overview about the chosen algorithms and procedures, and describes why they are suitable for this work.

3.1 Decision trees

The first machine learning method that is introduced is decision trees. The method has been evaluated, accepted and regularly applied in many fields for rule generation. One reason for its success is that the generated rules are human readable and thus is useful for a later analysis by human experts, which might not be the case with other data-mining methods. Also a complete description of the decision tree algorithm is freely available in literature (Quinlan, 1988).

3.1.1 Divide and conquer

The following example is widely used throughout the literature (Witten and Frank, 1999) to explain the concept of decision trees. The data shown in Table 3.1 consists of a training set for playing golf outside. It consists four attributes ("Outlook", "Temperature", "Humidity", and "Windy") and two classes ("stay inside", "go outside").

	Outlook	Temperature	Humidity %	Windy	Class
1	sunny	80	90	true	stay inside
2	rain	65	70	true	stay inside
3	sunny	72	95	false	stay inside
4	rain	68	80	false	go outside
5	overcast	83	78	false	go outside
6	sunny	75	70	true	go outside
7	overcast	64	65	true	go outside
8	sunny	69	70	false	go outside
9	rain	71	80	true	stay inside
10	overcast	81	75	false	go outside
11	rain	75	80	false	go outside
12	sunny	85	85	false	stay inside
13	overcast	72	90	true	go outside
14	rain	70	96	false	go outside

Table 3.1: This table represents a training set on which mining should generate a decision tree. Unknown instances should be classified to "go outside" and "stay inside" prediction on the basis of the weather preconditions outlook, temperature, humidity, and the presence or absence of wind.

The idea of a divide-and-conquer algorithm is to split the data table into subsets based on the different attributes. If the first split were based on the attribute outlook we create the subsets shown in the Table 3.2:

	Outlook	Temperature	Humidity	Windy	Class
1	sunny	80	90	true	stay inside
12	sunny	85	85	false	stay inside
3	sunny	72	95	false	stay inside
8	sunny	69	70	false	go outside
6	sunny	75	70	true	go outside

	Outlook	Temperature	Humidity	Windy	Class
9	rain	71	80	true	stay inside
11	rain	75	80	false	go outside
4	rain	68	80	false	go outside
2	rain	65	70	true	stay inside
14	rain	70	96	false	go outside

	Outlook	Temperature	Humidity	Windy	Class
5	overcast	83	78	false	go outside
7	overcast	64	65	true	go outside
10	overcast	81	75	false	go outside
13	overcast	72	90	true	go outside

Table 3.2: Subsets based on a split on the attribute outlook with the values "sunny", "rain", and "overcast".

It is easy to derive directly from the last subset that the decision to play outside is always true if the outlook is overcast. From this subset the first level of a decision tree as shown in Figure 3.1 can be created:

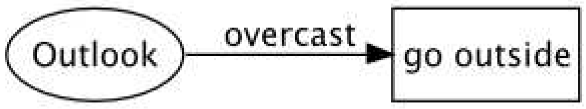


Figure 3.1: Decision tree containing the first level based on the attribute "overcast".

Since there are still mixed classes in the other subsets we have to split again the subsets until we end up with subsets without mixed classes. By splitting the second subset based on humidity we would end up with the following subsets shown in Table 3.3:

	Outlook	Temperature	Humidity	Windy	Class
8	sunny	69	70	false	go outside
6	sunny	75	70	true	go outside

	Outlook	Temperature	Humidity	Windy	Class
1	sunny	80	90	true	stay inside
12	sunny	85	85	false	stay inside
3	sunny	72	95	false	stay inside

Table 3.3: Subsets based on a split on the attribute outlook with the value of "overcast".

Now both subsets contain cases from a single class. Another test on humidity with outcomes dependent up on whether the humidity is less than 85 % or greater than 85 % can be used to generate the next level of the tree as shown in Figure 3.2.

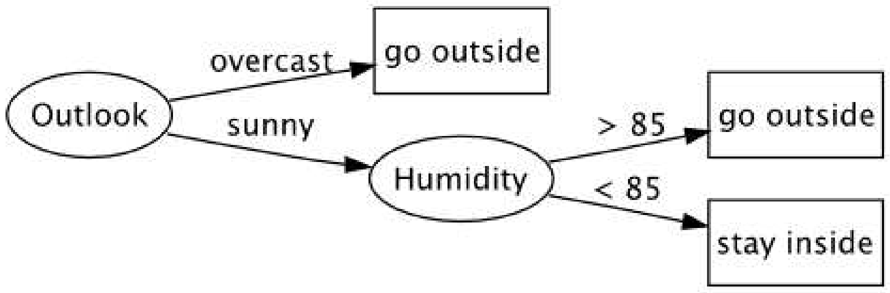


Figure 3.2: Decision tree containing the second level based on the attribute "sunny".

Splitting the subset which contains all instances of the example outlook rain creates the following subsets:

	Outlook	Temperature	Humidity	Windy	Class
11	rain	75	80	false	go outside
4	rain	68	80	false	go outside
14	rain	70	96	false	go outside

	Outlook	Temperature	Humidity	Windy	Class
9	rain	71	80	true	stay inside
2	rain	65	70	true	stay inside

Table 3.4: Subsets based on a split on the attribute outlook with the value of "rain".

The next level of the decision tree as shown in Figure 3.3 can now be created, while using another test on the attribute with the value "windy".

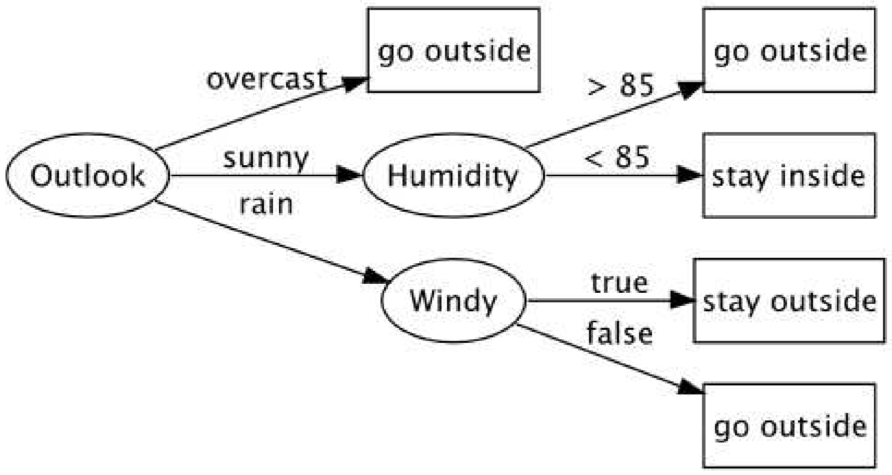


Figure 3.3: Decision tree containing the third level based on the attribute "windy".

3.1.2 Choosing the best attribute

The aim is to find the shortest decision tree that is completly consistent with the training set. So why not simply use a brute force method and try all possible decision trees to find the shortest one ? Unfortunately, finding the smallest decision tree consistent with a training set is a NP-complete problem (Hyafil, 1976). Therefore most decision tree construction methods are based on a heuristic approach which is normally based on non-backtracking greedy algorithms. One famous method which was originally introduced by Earl Hunt is to use a concept from information theory using the information gain.

Consider selecting one case at random from a set S of cases and announcing that it belongs to some class C_i. The message has the probability

$$\frac{freq(C_i, S)}{|S|}$$

and the information that it conveys is

$$-log_2\left(\frac{freq(C_i, S)}{|S|}\right) \text{bits}$$

To find the expected information from such a message we sum over the classes in proportion to their frequencies in S:

$$info(S) = -\sum_{j=1}^{k} \frac{freq(C_j, S)}{|S|} \cdot log_2\left(\frac{freq(C_j, S)}{|S|}\right) \text{bits}$$

After partitioning the set S in accordance with the n outcomes of a test X the expected information requirement can be found as the weighted sum over the subsets

$$info_X(S) = \sum_{i=1}^{n} \frac{|S_i|}{|S|} \cdot info(S_i)$$

The quantity

$$gain(X) = info(S) - info_X(S)$$

measures the information that is gained by partitioning S in accordance with the test X. One criterion to select a test named the ”gain criterion” is to take the one which maximizes this quantity.

Going back to the example, it is easy to calculate that

$$info(S) = -\frac{9}{14} \cdot log_2\left(\frac{9}{14}\right) - \frac{5}{14} \cdot log_2\left(\frac{5}{14}\right) = 0.94 \text{ bits}$$

Comparing the test for "Outlook" with the one for "Windy" gives the following equation

$$\begin{aligned}
info_{Outlook}(S) &= \tfrac{5}{14}\cdot\left(-\tfrac{2}{5}\cdot log_2\left(\tfrac{2}{5}\right)-\tfrac{3}{5}\cdot log_2\left(\tfrac{3}{5}\right)\right) \\
&+ \tfrac{4}{14}\cdot\left(-\tfrac{4}{4}\cdot log_2\left(\tfrac{4}{4}\right)-\lim_{x\to 0}\left(\tfrac{x}{4}\cdot log_2\left(\tfrac{x}{4}\right)\right)\right) \\
&+ \tfrac{5}{14}\cdot\left(-\tfrac{3}{5}\cdot log_2\left(\tfrac{3}{5}\right)-\tfrac{2}{5}\cdot log_2\left(\tfrac{2}{5}\right)\right) \\
&= 0.694 \text{ bits} \\
gain(Outlook) &= 0.94 \text{ bits - } 0.694 \text{ bits} = 0.246 \text{ bits} \\
info_{Windy}(S) &= \tfrac{6}{14}\cdot\left(-\tfrac{3}{6}\cdot log_2\left(\tfrac{3}{6}\right)-\tfrac{3}{6}\cdot log_2\left(\tfrac{3}{6}\right)\right) \\
&+ \tfrac{8}{14}\cdot\left(-\tfrac{6}{8}\cdot log_2\left(\tfrac{6}{8}\right)-\tfrac{2}{8}\cdot log_2\left(\tfrac{2}{8}\right)\right) \\
&= 0.892 \text{ bits} \\
gain(Windy) &= 0.94 \text{ bits - } 0.892 \text{ bits} = 0.048 \text{ bits}
\end{aligned}$$

The gain was used in former implementations of the algorithm but this resulted in maximal fragmentation of the data set. Assuming there is a criterion X which is different in every instance the quantity $info_X(S)$ would equal 0 and the $gain(X)$ would be maximal. However, this criterion is completely useless for prediction purposes. To avoid this artifact in more recent implementations, the gain was normalized with the information inherent in the split:

$$split\ info(X) = -\sum_{i=1}^{n} \frac{|S_i|}{|S|} \cdot log_2\left(\frac{|S_i|}{|S|}\right)$$

The *split info* is maximal for ”middle sized” and minimal for very small or very large groups. This criterion is used to define the *gain ratio*:

$$gain\ ratio(X) = \frac{gain(X)}{split\ info(X)}$$

The gain ratio ought to be maximal, since it is best to have a large *gain* and a small *split info*. For the example this value is:

$$\begin{aligned}
split\ info(Outlook) &= -\tfrac{5}{14} \cdot log_2(\tfrac{5}{14}) - \tfrac{4}{14} \cdot log_2(\tfrac{4}{14}) - \tfrac{5}{14} \cdot log_2(\tfrac{5}{14}) \\
&= 1.577 \text{ bits} \\
gain\ ratio(Outlook) &= \frac{0.246}{1.577} = 0.156 \\
split\ info(Windy) &= -\tfrac{6}{14} \cdot log_2(\tfrac{6}{14}) - \tfrac{8}{14} \cdot log_2(\tfrac{8}{14}) \\
&= 0.985 \text{ bits} \\
gain\ ratio(Windy) &= \frac{0.048}{0.985} = 0.049
\end{aligned}$$

This means that in this case *gain ratio*(*Outlook*) will be used to divide the set with a test for "Outlook".

3.1.3 Confidence score

The C 4.5 algorithm produces a decision tree for each given situation, regardless of whether or not there is a strong evidence. For all the cases where no clear pattern in the given attributes is detected, the quality of the predictor is too low to be significant. Low-quality predictions should not be applied, and for this reason a confidence score could be used as an estimated error rate for the tree. A common approach is to come up with an error estimate as the standard verification technique. Some data of the original given data set could be held back and afterwards used as an independent test set to estimate the error of the tree, (Witten and Frank, 1999). This method is called reduced-error evaluation. It suffers from the disadvantage that the actual tree is based on less data. An alternative approach is to try to make some estimate of the error based on the training data itself. The following method is taken from the original Spearmint implementation. The procedure tries to derive a score by counting, using the number of true positive (TP) and false positive (FP) examples in the training set. The corresponding formula calculates the following value of likelihood: Given the number of true positive and false positive examples it calculates which rules lie above a given threshold in 95% of all cases (Kretschmann et al., 2001).

$$\begin{aligned}
z &= 1.96 \quad \text{(Constant for 95\%)} \\
n &= TP + FP \\
p &= Precision = \frac{TP}{TP + FP} \\
Score &= \frac{p + \frac{z^2}{2n} - z * \sqrt{\frac{p}{n} - \frac{p^2}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}}
\end{aligned}$$

3.2 Finite state machines

Protein sequences are represented as strings of characters, which makes it easy to apply to them computational methods for pattern detection. Signature hits such as protein domains and functional sites have become a common method for the prediction of protein function. For this reason, several recognition methods have evolved to address different sequence-analysis problems during the last decade. Regular expressions, hidden Markov models and sequence profiles have been applied in many signature database such as Prosite (Hulo et al., 2006), Pfam (Bateman et al., 2004) or InterPro (Biswas et al., 2002) for gaining a better understanding of uncharacterized proteins.

The underlying concept of these predictive methods is in most cases similar. On an abstract level regular expressions, hidden Markov models and profiles are specialized cases of finite state machines, where states and transitions are used to find a desired pattern, (Cortes et al., 2004). Diagnostically, all of these models have different areas of optimum application owing to different strengths and weakness. Regular expressions are conceptually straightforward to apply, but they cannot be used to find patterns with a stochastic structure. Hidden Markov models use a probabilistic approach which can deal with stochastic behaviour but the model needs to be optimized to a specific application, which makes its creation difficult. As an alternative to these approaches, weighted finite state machines (FSMs) can be used to detect signature hits in protein sequences, and by adding cost or weights to the transition it is also possible to handle stochastic behaviour with them, (Mohri, 1997).

The following chapter shows the basic principles of FSMs and how in particular a weighted FSM can be used for pattern recognition in bioinformatics. It is necessary to show how the algorithms are working in detail to apply them to the detection of patterns in sequences.

3.2.1 Acceptor

A finite state machine (FSM) is an abstract model which contains states, transitions and actions. The states in the FSM store information about the input data called input symbols. A transition indicates a state change and is normally described by a condition that is needed to be fulfilled to enable the transition, (Hopcroft and Ullman, 1979). The simplest model of a FSM is an acceptor. From a mathematical point of view a non-deterministic acceptor is defined as a quintuple (Q, τ, F, $\mathit{\Gamma}$, δ):
Q is a finite set of states,
τ is the set of initial states,
F is the set of final states,
$\mathit{\Gamma}$ is the input alphabet,
δ is the state transition function δ : Q x $\mathit{\Gamma}$ to Q.

The state transition function is defined as the cross-product between the set of all states Q and the input alphabet $\mathit{\Gamma}$ and defines the transitions between the states in the acceptor. An acceptor can be visualized using a state transition diagram as shown in Figure 3.4 where the displayed acceptor contains one initial state marked with a bold circle and one final state marked with a double circle. The input alphabet is based on "A", "G", "T" and "C" , where the state transition function defines the transition from one state to the other.

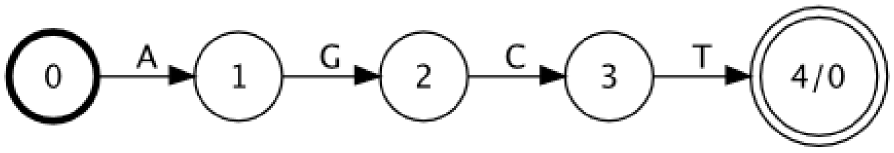


Figure 3.4: 3 Acceptor which accepts the input sequence "AGCT" using 5 states and four transitions.

3.2.2 Transducer

Transducers are extensions of normal acceptors. A transducer is a FSM that also generates output symbols based on a given input symbol. The output generation is called an action and can be used to define a mapping between two different types of alphabets. Transducers can be used to map different information resources together where each resource is represented by its own alphabet. This is helpful for several applications, e.g. in speech recognition to provide a mapping between voice and words.

A nondeterministic transducer is defined as a septuple (Q, τ, F, $\mathit{\Gamma}$, θ, δ, ρ):
Q is a finite set of states,
τ is the set of initial states,
F is the set of final states,
$\mathit{\Gamma}$ is the input alphabet,
θ is the output alphabet,
δ is the state transition function $\delta :$ Q x $\mathit{\Gamma}$ to Q,
ρ is the output function $\rho :$ Q x θ to Q.

The difference from the acceptor is that in this case an input alphabet and a output alphabet are defined where the output function is used by the state transition to define the actual mapping of the transducer. The transducer shown in the state transition diagram in Figure 3.5 contains 5 states with one initial state and one final state. Each transition of the transducer emits a symbol once a transition is reached. The input alphabet $\mathit{\Gamma}$ and the output alphabet θ are in this case similar and contain the characters "A", "G", "T" and "C" .

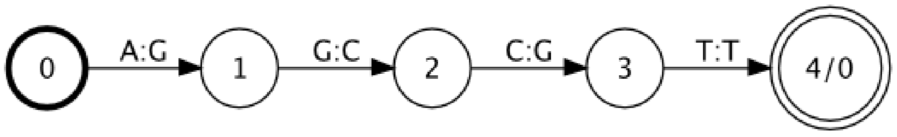


Figure 3.5: Transducer which accepts the input sequence "AGCT" and produces the output sequence "GCGT".

3.2.3 Weighted acceptor

FSMs as described before can handle exact pattern matching, but unfortunately it is not possible to handle stochastic behaviour with them. If the preferred pattern which needs to be detected varies slightly, it is impossible to detect it by a normal acceptor. For this reason weighted FSMs were introduced, (Mohri, 1997). Weighted FSMs are a specialized case of FSMs where weights are added to the transition. These weights work in this context as negative logarithmic probabilities. The advantage of adding weights or costs to a transition is that this provides a way to prefer one transition over another.

A nondeterministic weighted acceptor is defined as a septuple (Q, τ, F, Γ, δ, λ, ω):
Q is a finite set of states,
τ is the set of initial states,
F is the set of final states,
Γ is the input alphabet,
δ is the state transition function δ : Q x Γ to Q,
$\lambda \in R_{+}$ is the initial weight,
ω is the weight function: F to R_{+}.

The weighted acceptor contains in addition to the normal transducer a weight function where specific weights can be defined for each transition, where the number of the weight has to be part of the the set of real numbers (R_{+}). It contains also an initial weight function where a default weight is defined for each transition. Each transition from the transducer in Figure 3.6 has a weight added to the input symbol.

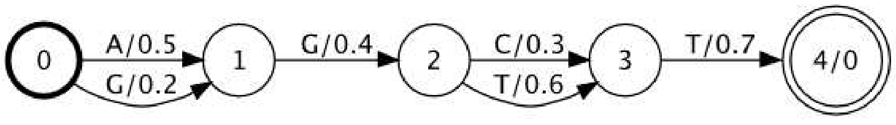


Figure 3.6: Example of a weighted acceptor.

3.2.4 Weighted transducer

Weighted transitions can also be applied to a transducer, the difference being that the weight is added to a transition which contains both an input and an output symbol. The output symbol then gets emitted based on the weight of the transition. So in this case a weighted transducer provides a mapping between two different alphabets based on a stochastic behaviour defined by the weights. This can be helpful to modeling complex relationships between different data sources, e.g. in defining the probability of to which word a specific voice is related.

A non-deterministic weighted transducer is defined as a nonatuple (Q, τ, F, $\mathit{\Gamma}$, θ, δ, ρ, λ, ω):
Q is a finite set of states,
τ is the set of initial states,
F is the set of final states,
$\mathit{\Gamma}$ is the input alphabet,
θ is the output alphabet,
δ is the state transition function $\delta :$ Q x $\mathit{\Gamma}$ to Q,
ρ is the output function $\rho :$ Q x θ to Q,
$\lambda \in R_+$ is the initial weight,
ω is the weight function: F to R_+.

A weighted transducer is very similar to a normal transducer except that a weight function and an initial weight is defined for each transition. The weighted transducer shown in Figure 3.7 contains a weight for each transition which defines the probability that an output symbol is emitted based on a given input alphabet.

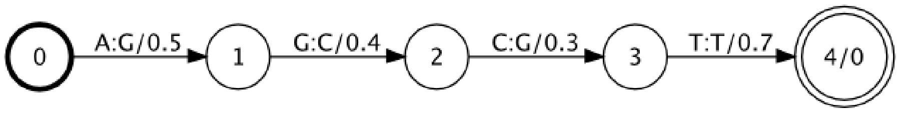


Figure 3.7: Transducer containing weighted transitions.

3.2.5 Operations

Several standard operations are defined for FSMs. The following operations are essential when working with FSMs, (Hopcroft and Ullman, 1979). These operations behave in the same way with unweighted and weighted FSMs, except that in the weighted case the weights of the transition will also be affected, (Mohri, 1997).

Composition

The composition is a key operation on transducers and used to create complex weighted transducers from simpler ones. With the composition operation two finite state machines can be combined into a new one. The output symbol of the first FSM is mapped to the input symbol of the second FSM and in the case that the transitions are weighted, the weights of the transitions will be summed up. Composition of weighted acceptors is a special case of composition of weighted transducers which corresponds to the case where the input and output symbols of the transitions are identical. The following example in Figure 3.8 combines two transducer using composition into a new transducer. The first transducer contains a transition from state 0 to state 1 with the weight of the input symbol "A" and the output symbol "T". This output symbol is mapped to the transition from state 0 of the second transducer to the input symbol "T".

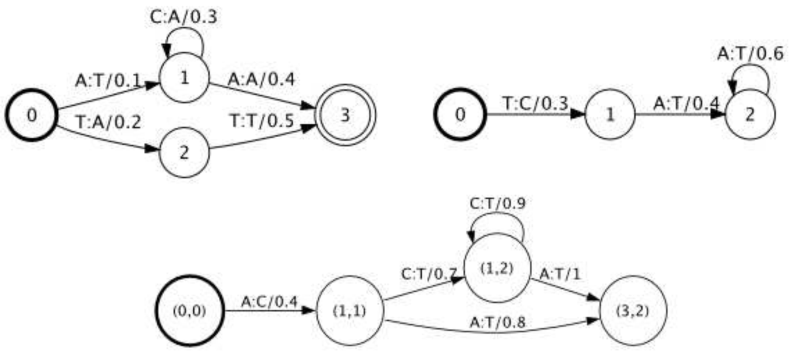


Figure 3.8: Combining two transducers into a new one using the composition operation.

Minimization

The minimization operation can be used to reduce the size of the states of a FSM, (Hopcroft and Ullman, 1979). From the definition a weighted FSM is minimal if there exists no other weighted FSM with a smaller number of states realizing the same function. Also, two states of a weighted FSM are equivalent if exactly the same set of strings with the same weights label paths from these states to a final state. Thus, two equivalent states of a deterministic weighted automaton can be merged without affecting the function realized by that weighted finite state machine. This is very useful because it allows the size of the intermediate compositions to be reduced, thus saving on computing-time and memory. The example in Figure 3.9 uses minimization to reduce the number of states in a weighted acceptor. Note that the relationship between state 2 and state 3 is exactly the same as that between state 4 and state 5. These states can be replaced by one state as shown in the resulting transducer.

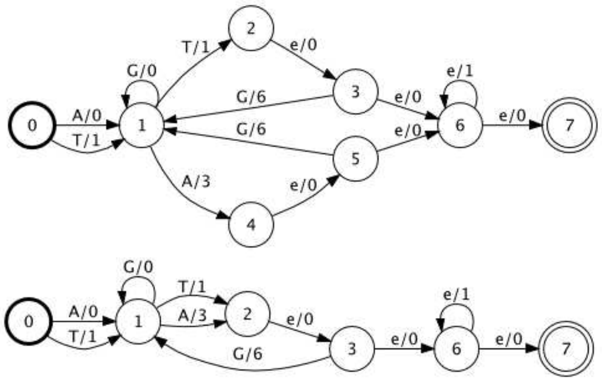


Figure 3.9: The number of states in a weighted acceptor is reduced using the minimization operation.

Finding the shortest path

Finding the shortest path is a very important operation when using weighted FSMs, (Hopcroft and Ullman, 1979). The idea is to find the "best" solution in the set of possible solutions represented by a a FSM, which is equivalent to finding the highest probability path through a network. The algorithm starts from an initial state and chooses only those transitions with the lowest weight, while all other transitions and their corresponding states are removed. The application of the shortest path algorithm is shown on a weighted acceptor in Figure 3.9 where the acceptor containing similar behaviour depending based on the input symbol and the weight. State 0 contains two transitions, one with an input symbol "A" with a weight of 0.5 and another one with an input symbol "T" and a weight of 0.3. In the resulting acceptor the transition "T" is removed. States 1 and 2 gets also removed because there are no transitions pointing to them.

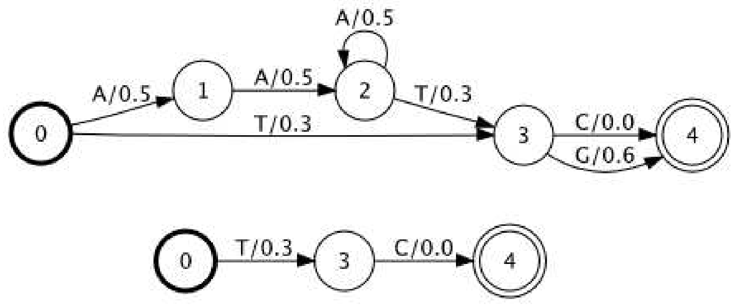


Figure 3.10: Example of finding the shortest path in a weighted acceptor, by replacing similar behaviour of the acceptor with a smaller set of states.

Pruning

Pruning is a method of removing all transitions and the corresponding states of a weighted FSM higher than a specific threshold. A threshold can be defined, and the algorithm visits each transition and removes it if the weight of the transition is higher than the threshold. This method is useful for optimizing a weighted FSM and remove all states and transitions where the possibility of their execution is not very likely. The weighted acceptor shown in Figure 3.11 is reduced using the pruning operation with a threshold value of 5. The resulting acceptor contains only transitions where the corresponding weights are below 5. All other transitions and the connected states are removed.

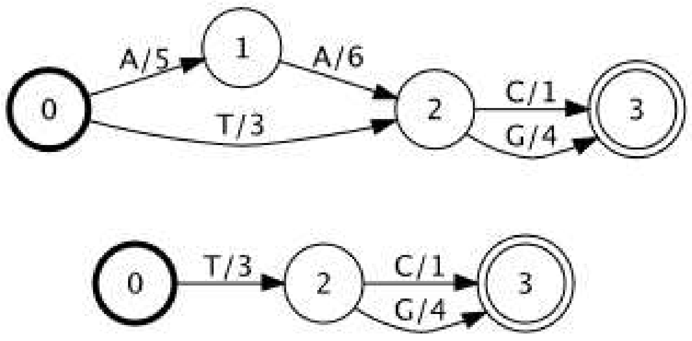


Figure 3.11: Pruning of a weighted acceptor. The resulting acceptor contains only transitions where the corresponding weights are below 5.

3.2.6 Building a predictive model

The following example shows how weighted FSMs are used to create a predictive model for pattern recognition in nucleotide sequences. Three different training-sequences contain a similiar pattern in the sequence, and this pattern should be recognized by a weighted FSM in an unknown sequence. A gap penalty transducer is used to introduce gap penalties. The input alphabet for the FSM defines a representation of the four nucleotides. First, all sequences have to been converted into weighted acceptors. Each possible nucleotide is represented as a weighted transition. The input symbol of the transition represents the nucleotides and an additional epsilon transition is added to introduce gap penalties. The acceptor in Figure 3.12 shows the unweighted case for representing a four-nucleotide sequence.

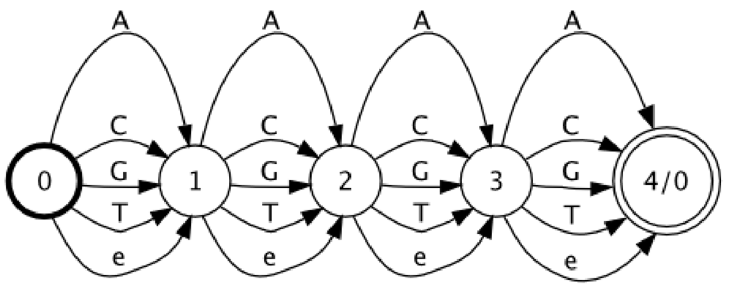


Figure 3.12: Acceptor in the length of a four-nucleotide sequence.

To create an acceptor that represents the actual sequence, the acceptor needs to be modified to specify which nucleotide sequence should be present. The following example in Figure 3.13 shows three acceptors using weighted transitions which represent the sequences "ATGC", "ATTC" and "ATCG" . For the presence of a nucleotide a weight of -10 was added to the transition, while otherwise the default weight of 1 is used.

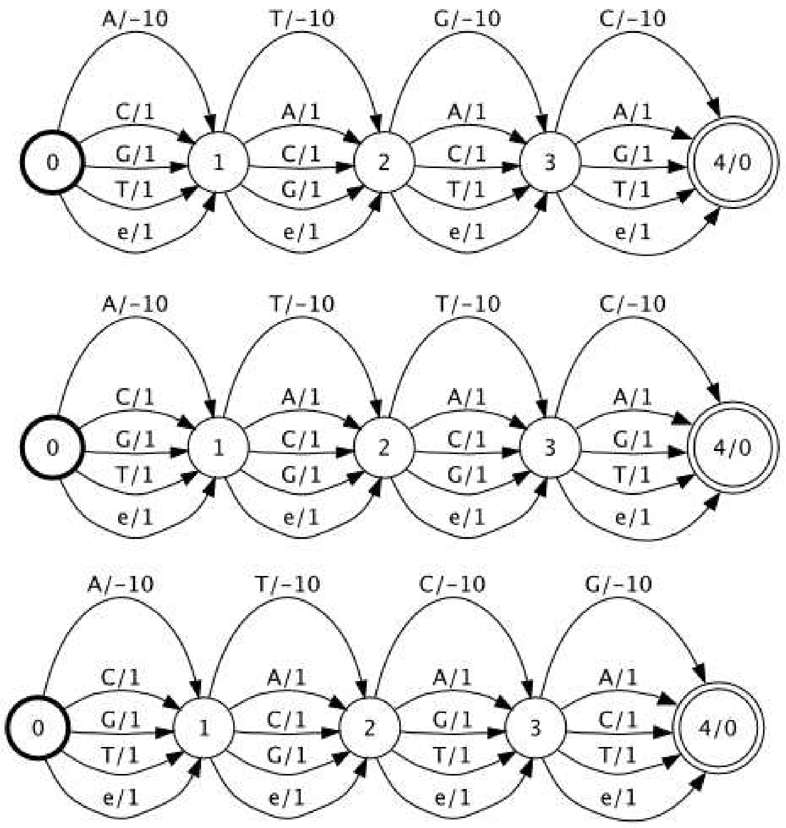


Figure 3.13: Three acceptors representing the the sequences "ATGC", "ATTC" and "ATCG" .

These three acceptors can now be combined to one acceptor using the composition operation. The result is a new acceptor shown in Figure 3.16 which contains the patterns of all three sequences. The resulting acceptor was pruned, to avoid uneccasary complexity.

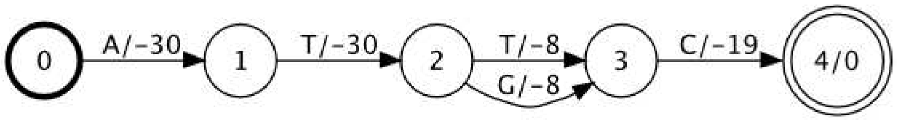


Figure 3.14: Acceptor which contains all 3 example sequences at once.

To add additional behaviour such as gap penalties, the given sequence acceptor needs to be combined with a gap penalty transducer. The gap penalty transducer in Figure 3.15 contains weighted epsilion transitions defined by the nucleotide input alphabet. An epsilion transition is executed based on its input weight and indicates that a gap can be inserted at this position.

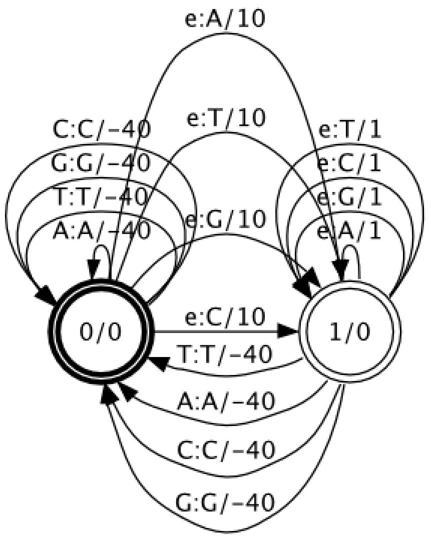


Figure 3.15: Transducer introducing gap penalties based on a nucleotide alphabet.

A predictive model can be created using the composition operation between the sequence representation and the gap penalty transducer. The result is a more complex transducer shown in Figure 3.16 which contains the patterns of the training-sequences and the gap penalties stored in the weights of the transducer. This transducer can now be used to query for the pattern in an unknown sequence. Again, to avoid unnecessary complexity this transducer was also pruned and afterwards minimized.

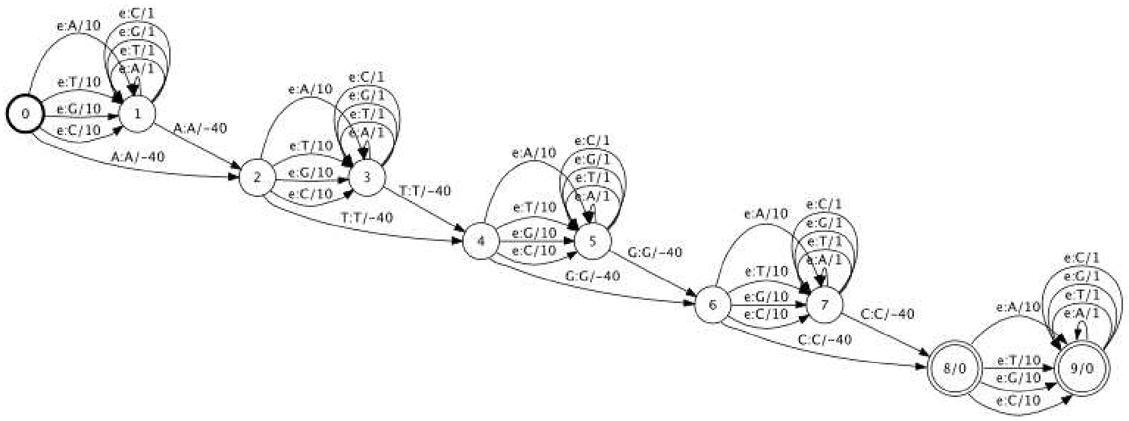


Figure 3.16: Result transducer containing the sequence patterns and the gap penalties.

An unknown query sequence need to be converted to an acceptor. No weights needs to be added, because there is no uncertain behaviour in the query sequence. Figure 3.17 shows the sequence "ATGC" as an unweighted acceptor.

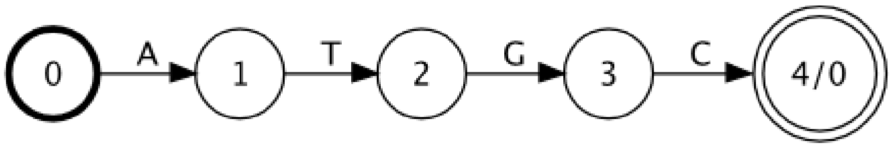


Figure 3.17: Query sequence "ATGC" represented as an unweighted acceptor.

A new transducer needs to be created combining the acceptor containing the sequence from Figure 3.17 with the transducer that contains the predictive model using again the composition operation. Then the shortest path operation is applied to find the best alignment between the acceptor and the transducer that contains the predictive model. The result is shown in Figure 3.18. The values of the weights on the transitions describe the level of similarity between the predictive model and the query sequence.

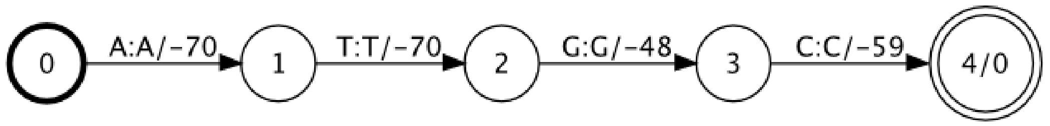


Figure 3.18: Result transducer after applying the acceptor against the predictive model.

3.3 Summary

Two methods have been shown to be suitable for the two stage approach. Firstly, the C 4.5 decision tree algorithm, described in the previous chapter, which creates trees of rules on the basis of information gain. The algorithm is used to generate protein annotations in TrEMBL by the Spearmint system (Kretschmann et al., 2001). Spearmint is able to predict the presence of a functional comment type, which describes the functional behaviour of protein. The algorithm scales quadratically with the number of instances to learn on, but to reduce computational costs the Swiss-Prot dataset was divided into small sections based on their InterPro families (Biswas et al., 2002), upon which the algorithms learn. However, the method could be extended to predict the presence of a functional annotation, but since every functional annotation has a corresponding position on the sequence, another method is needed to analyse the sequence.

Secondly, FSMs allow efficient and powerful modelling of biological problems. On an abstract level, acceptors represent a sequence of input symbols, while transducers encode a mapping between input and output sequences. Weights such as match or mismatch probability can be assigned to each transition. Simple models like regular expressions, and also more complex ones like hidden Markov models and support vector machines, can be modeled because they are specialized cases of finite state machines (Cortes et al., 2004). Thus, they provide a generic concept for modelling complex sequence data and algorithms using a set of algebraic operations. They deliver a framework with which predictive methods can be combined conveniently to produce more powerful predictors.

4 Implementation

The application developed here is a Java (Goslin, 1988) based system using a two-stage approach. The two algorithms described in the previous chapter are used to predict the sequence position of a functional site and the related annotation in the UniProtKB (Wu et al., 2006). The requirements for the application are to measure the performance of the two combined algorithms and to evaluate whether the results are suitable for integration into the automated annotation pipeline of the UniprotKB. Normally it is not a trivial task to measure the performance of a predictive algorithm. The system needs to be able to perform a cross-validation against the Swiss-Prot part of the UniProtKB. First of all, a small prototype was developed to test the accuracy and potential of the predictions. The results of the evaluation phase were promising, and the decision was made to implement a system in an object-oriented design. The final implementation of the system was divided into four parts. The first part is a module that provides the algorithms described above, which are necessary for the combined prediction approach. The second part is the implementation of a workflow engine to model the behaviour of the data-mining run. As part of this module, infrastructure was developed to load and analyze the proteins. The third part was a backend module to provide the application with the protein-related data and a relational database to store the results of the predictions. The final part is a graphical web user interface that allows internal and external users to browse the generated data and give feedback on the quality of the results. The following chapter gives an overview of the software design of the system. The chapter explains the design issues and design patterns which were used to provide a flexible and easy-to-use software system.

4.1 Algorithm module

The algorithm module contains separate implementations of the algorithms needed for the application. An implementation of the C 4.5 decision tree algorithm was developed to predict the presence of a functional site on a protein. In the sequence-analyzing step, the external finite state machine library from AT&T has been used to find the exact position of the functional site.

4.1.1 Decision trees

The C 4.5 Decisiontree algorithm shown above is used in the first stage of the application to predict the presence of a functional site. Several free open-source implementations of this algorithm are available. As an example, the weka machine learning software package (Witten and Frank, 1999), which is embedded in the UniprotKB automated annotation pipeline to provide regular annotations on proteins. Unfortunately there are limitations to this package which make it unusable for this application. The generated decision tree is only available as textual output, so there is no possibility of accessing the internal structure. However, internal access is needed to extract the positive examples used for building the tree. These examples are needed to extract the protein sequences for the sequence-analysis step. To this end, an implementation of the C 4.5 algorithm in Java has been created and satisfies all of the desired criteria. The chief advantage of this approach is a clean tree structure which is easy use. The class diagram in Figure 4.1 gives an overview over the design of the implementation. The core design of the algorithm contains four classes and can be divided into two parts. Two classes provide the tree structure, and two classes provide the heuristic algorithm.

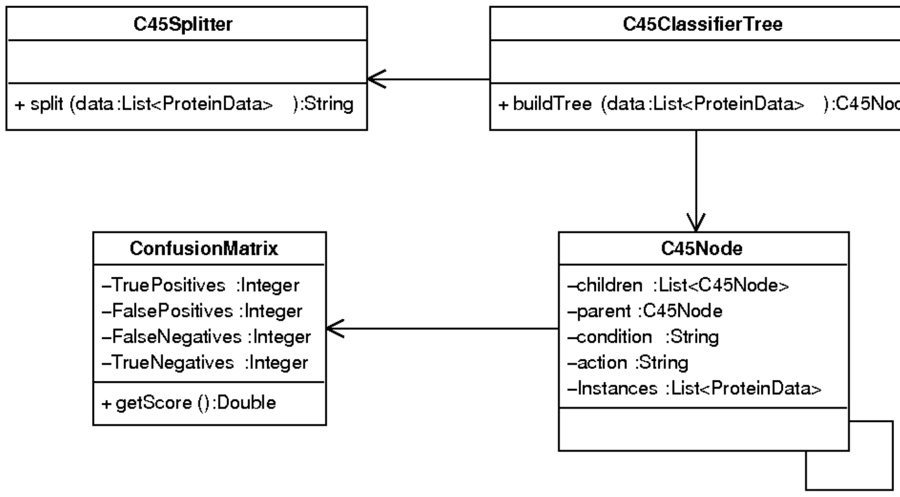


Figure 4.1: Class diagramm of C 4.5 decision tree implementation.

The main interface of the implementation is the class C45ClassifierTree. The interface takes a dataset of protein objects as an input parameter and returns a Decisiontree. The Decisiontree is composed of class C45Nodes, where each node contains a parent node and several children nodes. Each of these nodes contains all positive learning examples, a condition, and an action. The condition describes the attribute of the node, e.g. a specific signature hit. The action describes the desired result. Each node provides basic statistics about the splitting process using the class ConfusionMatrix. The confusion matrix is a datatype taken from the original C 4.5 implementation (Quinlan, 1988) and contains the number of true/false and positive/negative examples. The splitting process is done by the class C45Splitter that separates instances from the dataset based on the information gain of their attributes. The attribute with the highest information gain is returned as the condition for the node. This process continues recursively until all instances in the branch have the same value or there are no more attributes left to be separated. For a more detailed description of the algorithm see in chapter 2.

The decision tree shown in Figure 4.2 shows an example of a decision tree created using signature hits and taxonomy information about a protein to predict an active site. Each node has a condition, and the leaf node contains an action which predicts an active site annotation. Also each node contains a confusion matrix where the true positives, false positives, false negatives, and true negatives are counted. The rules is described as followed: If the protein contains the Prosite hit 010701, the Pfam hit 00198, and its taxonomy is not Homo Sapiens then the decision tree predicts an active site on this protein. An unknown protein gets classified by this tree visiting each of its nodes.

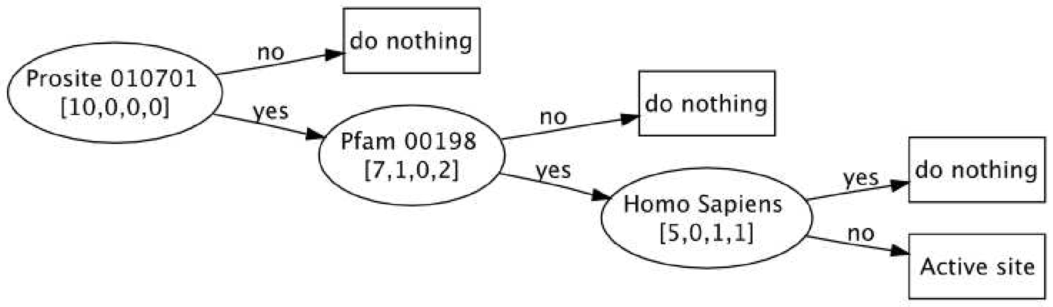


Figure 4.2: Decision tree using signature hits and taxonomy to predict the present of a active site annotation. Each node contains a confusion matrix where the true positives, false positives, false negatives, and true negatives are counted.

4.1.2 Weighted finite state machines

Weighted finite state machines (FSMs) are employed for the sequence analyzing step, to find the exact position of a functional site. The FSM library from AT&T is used for the implementation of the algorithm. The library provides an efficient application programmer interface (API) to the corresponding algorithms to build and manipulate FSMs. The potentials of the library to model sequence related algorithms was introduced in the tutorial *Weighted Finite-State Transducers in Computational Biology* at the 13th Annual International Conference on Intelligent Systems for Molecular Biology. The library itself is written in the C programming language (Kernighan and Ritchie, 1988) and available as binary distribution for several plattforms. The library provides a simple command line interface which is used for building, combining, and optimizing weighted finite-state acceptors and transducers. The data-mining application is completly written in Java so the application cannot communicate directly with the FSM library. Thus the communication between the Java application and the command line interface is handled by a converter class which executes a system call to the FSM library.

Class FSMConverter

The class FSMConverter converts a protein sequence into a FSM representation. An amino acid alphabet is used for the input and output symbol of the weighted FSM. The class generates afterwards all the necessary textfiles for the FSM library and executes a system call to employ the FSM library on the command line interface. The results of the FSM library is represented as a textfile that is parsed and handed back to the application. In this way a collection of the essential methods for working the AT&T FSM library are provided to the application. Weighted acceptors and transducer can be created, on a given input alphabet, and they can be composed, minimized, pruned, and applied to the best path algorithm.

4.2 Workflow module

The workflow of a data-mining routine is a complex behaviour. Several tasks have to be executed and the ordering of these tasks can change for each single run, so the environment needs to be flexible. The normal usecase is that a data set of proteins is loaded from an external datasource where all important information about a protein for the data-mining process is stored. A classifier is created based on this information. This classifier is then applied to the protein objects to make a prediction. At the end the created predictions are saved in a relational database.

This workflow can vary for each data-mining run. Sometimes proteins need to be filtered out from instances which are not suitable for a specific datamining run such as if there are enourmous protein like fragments. For this reason it must be possible to introduce new data-processing steps easily. The state pattern (Gamma, 1997) addresses exactly this problem. The pattern provides a way of changing the behaviour of an object depending on its state, and the different states of the object can be implemented separately. The implementation of this pattern is shown in the class diagram 4.3. The class StateMachine can have a number of internal states. The interface "State" defines the behaviour of a single step of the application which is implemented by concrete states. A product which contains the data will be delegated by the statemachine to the states in a specific order for handling of the processing steps.

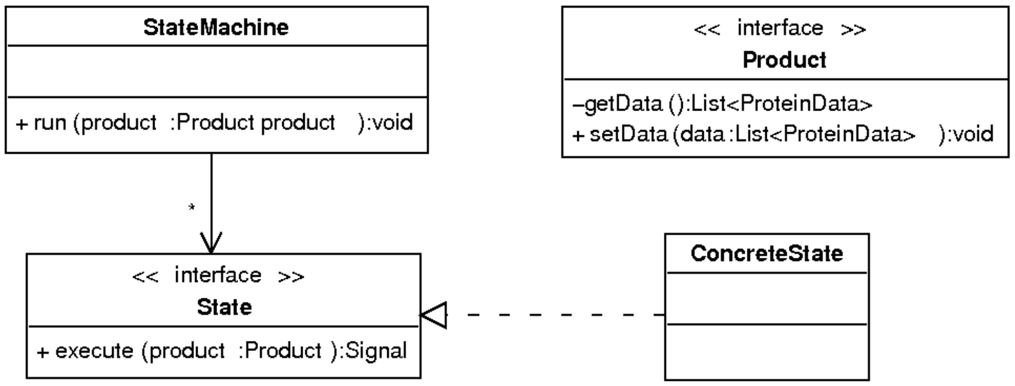

StateMachine
+ run (product :Product product):void
<< interface >>
Product
–getData ():List<ProteinData>
+ setData (data :List<ProteinData>):void
*
<< interface >>
State
+ execute (product :Product):Signal
ConcreteState

Figure 4.3: Class diagram of the workflow module of the application.

A metaphor for this design pattern would be a conveyer belt in a factory. A product is driven through a factory and stops at different workers to be processed. The workers do not know much about the full product, having only very basic assumptions about it. The single worker is an expert in his own area, and he just adds his modifications and passes it to. Once the product has left a worker the management t decides where to send the product to. The benefit of this principle is that the workflow can be easily extended, by just including another worker at the conveyer belt. Each logical step in the application can be implemented as a separate state which gives lots of benefits to the design of the application. Also, the individual states of the application can be split into trivial subprocesses that can be optimized individually without paying attention to the whole process. The subprocesses do not depend on each other and can be reassembled easily to achieve improvements for the whole process. Each subprocess can be implemented and tested independently from the rest of the system and afterwards easily integrated.

Good object-oriented software design works in a very similar way. So how can this design be implemented in the data-mining application? Simply, every logical step of the data-mining process will be modelled as its own state which takes a data set of protein objects, performs an operation and then passes the set back to a state machine which delegates the data set of proteins to the next state. Based on this design pattern each logical step of the data-mining run was identified by a use-case and was implemented afterwards in a separated classes. The class diagramm Figure 4.4 gives an overview of the implemented states. Every state implements the Interface "State" and provides a method for executing which performs an operation on a data set.

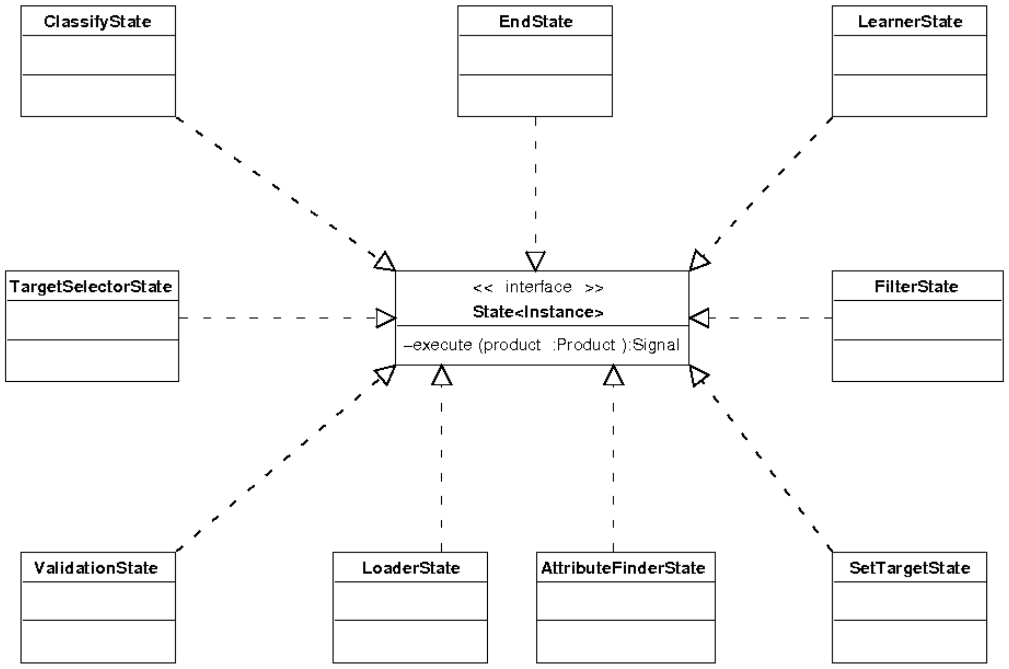


Figure 4.4: The class diagram shows the implemented states in the application. Each data-mining step of the application is implementing the interface "State".

Below the details of the implemented states are shown together with a description of their behaviour based on a use-case.

Class LoaderState

The loader state loads a data set of of protein objects from the backend module. The proteins are described based on the annotation within the Swiss-Prot section of the UniProtKB such as functional annotations, taxonomy or signature hits. To reduce computational costs the data set was loaded in small sections based on their InterPro families (Biswas et al., 2002).

Class FilterState

The filter state filters instances from the dataset which are not suitable for data-mining. This may be used to get rid of enourmous proteins such as hypothetical proteins which are not useful in a training set. Hypothetical proteins are usually not well annotated and a rather insecure source of data to be used for training set.

Class SplitterState

The splitter state splits a given data set of protein objects into small homogeneous sets to perform a cross-validation. As an example, a data set is split into five parts, of which four parts will be used as a training set to learn the algorithms, with the remaining fifth part, called the target set being tested. This procedure will be repeated five times, each time changing training and target sets. The number of sets in which the data sets will be split can be configured externally.

Class AttributeFinderState

The attribute-finder state extracts the attributes from protein objects of the data set. The data extraction process is configurable because not every attribute of a protein object is suitable for the data-mining process. As an example, a signature hit can be corrupt, so this case has to be excluded.

Class TargetSelectorState

The target-selector state chooses a target annotation where the classification algorithm should be learned. A decision tree can only generate one rule for one specific target. For example, to predict different instances of the active site annotation such as proton donor or binding site, individual decision trees need to be created for each annotation type.

Class C45LearnerState

The C 4.5-learner state takes a data set of protein objects including the attributes and the targets and uses the class C45ClassierTree from the algorithm modul to create a decision tree. This state delegates the data set to the decision tree implementation and returns the predictive model which can be used afterwards to predict the presence of a functional annotation. The created decision tree is employed on the target set to predict the presence of a functional annotation.

Class FSMLearnerState

A FSM is created based on the positive examples from the building procedure of the decision tree. All sequences containing the desired pattern are extracted and used to create a FSM. The FSM created by the FSM-learner state is applied against all protein sequences from the target set where the presence of a functional annotation is already predicted by a decision tree.

ValidationState

The presence of a functional annotation and the sequence-specific position gets evaluated against the original annotation, and will be written to the database in the backend modul. The evaluation distinguishes between the decision tree and FSM predictions in respect of the number of true/false positive/negative examples.

EndState

The data-mining run is finished if the end state is reached. The application is then terminated.

The implemented states provide a collection of reusable components for the automated annotation in a object-oriented way. New processing steps and procedures can be easily integrated into the whole process, while reconfiguring the states or implementing new states. The application was designed for the prediction of functional annotations on proteins, but the components can also be reused for other tasks in the automated annotation of the UniprotKB. The workflow and the order of the execution of the states is handled by the class StateMachine. This class can be configured programmatically to define the order in which the states are executed. The code-snippet in Figure 4.5 shows how the application can be configured programmatically:

```
LoaderState loader = new LoaderState();
AttributeFinderState finder = new AttributeFinderState();
TargetSelectorState selector = new TargetSelectorState();
C45ClassifyState classifier = new ClassifierState();
ValidatorState validator = new ValidatorState();

StateMachine stateMachine = new StateMachine();

stateMachine.setStartState(loader);
stateMachine.setTransition(loader, finder);
stateMachine.setTransition(finder, selector);
stateMachine.setTransition(selector, classifier);
stateMachine.setTransition(classifier, validator);
stateMachine.setTransition(validator, new Endstate());

Product product = new Product();
statemachine.run(product);
```

Figure 4.5: Programmatic configuration of the workflow of a data-mining run.

4.3 Backend module

The backend module provides the application with protein objects. A protein object is an object-oriented data model representation of an entry from the UniprotKB. It contains all information extracted from UniProtKB flatfile entry where proteins are described. For performance reasons the relevant data for the data-mining process was created using the yasp-parser which is an European Bioinformatics Institute (EBI) internal tool for parsing the UniProtKB flatfile format. The output of the parser was stored in a object-oriented domain model and stored on a hard drive instead of using the UniProtKB data warehouse. This way data is easier and quicker to access for the various runs of the application. For a normal data-mining run the time to read the relevant data from the data warehouse takes hours, while reading similar data from a hard drive is only a matter of minutes.

To reduce the computational costs of the mining process, the loaded data set was divided into small sections based on InterPro families. However, a mechanism had to been found to identify which proteins belongs to a specific InterPro family. For this reason a relational database was created to detect which proteins belong to a specific InterPro family. The database was also used to save the predictions of the data-mining run and make these available for a later evaluation. 4 tables have been identified and implemented to store all information needed to fulfill the requirements.

The following tables were created in the schema and implemented in Oracle:

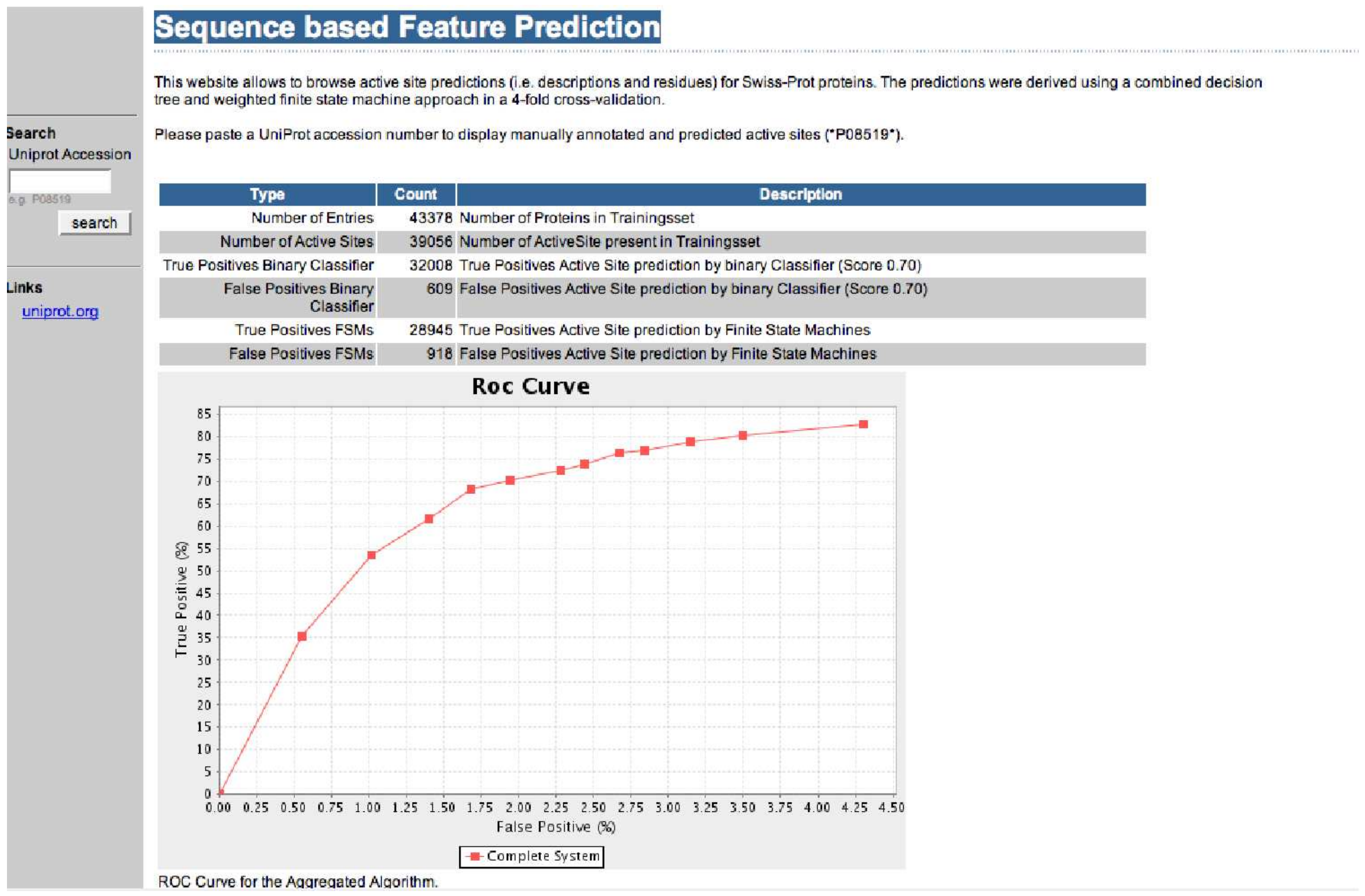
- Table interpro_entry: This table contains the information about an InterPro family. The InterPro family is described by name, a shortname and the corresponding InterPro accession number. Only InterPro families which contain proteins from the Swiss-Prot section of the UniProtKB have been stored.
- Table uniprot_id: This table contains the information about a protein. The primary UniProt accession number, the creation date and UniProt identifier have been extracted from UniProt entries and stored.
- Table intepro_protein: This table contains the relationships between a protein and an InterPro family. This is a many-to-many relationship, so this table links between the tables interpro_entry and uniprot_id.

- Table uniprot_prediction: This table contains all the predictions from the datamining run. It store each prediction that was performed on a specific protein entry. The textual description and sequence position is stored including the score of the decision tree from which the prediction was generated.

The database schema was added to query for the desired protein objects and to allow easy access to the generated data. It was not optimized in terms of efficiency and was designed to cover basic functionality. The table uniprot_prediction was later used to allow external user access to the data with a web interface.

4.4 Web module

The generated data needs to be visible for curators and external user to provide feedback about the quality of the data. The developed application is a separate system from the UniProtKB automated annotation pipeline and the results are not yet evaluated. Therefore the data cannot be integrated into UniProtKB website. However, it would be helpful to have a web-user interface to browse results. The developed web module is based on the model-view-control pattern (Gamma, 1997) and uses the JSP-based Open Source Framework Struts (Cavaness, 2004) for providing a visualization of the results. The application contains two different views. The first view, shown in Figure 4.6, provides an overview over the overall performance of the last data-mining run with detailed information about the data set. A statistical table shows the true positive and false positive rates separately for two different algorithms. Receiver operation characteristic (ROC) curves (Beck and Shultz, 1986) are provided for the aggregated algorithm and the decision trees. The entry view in Figure 4.4 provides access to predictions for single UniProtKB entries. The user can enter an UniProt identifier to receive individual predictions for an entry. The entry view shows the original functional site annotation from the UniProtKB and predictions from the data-mining run. The score value from the decision tree is added to each prediction. The web application provides also a way to browse the generated predictions immediately after their creation, and so can be used as a tool for monitoring the data-mining of the functional site predictions application. Wrong predictions, for example, because of wrong configurations of the application can be detected before the data-mining run is finished, and the application can be stopped and restarted. A version of the web application which contains the results of a 4-fold cross-validation of the Swiss-Prot section of UniProtKB version 7.0 is visible external at http://www.ebi.ac.uk/interpro/internal-tools/aa/siteprediction.



Sequence based Feature Prediction

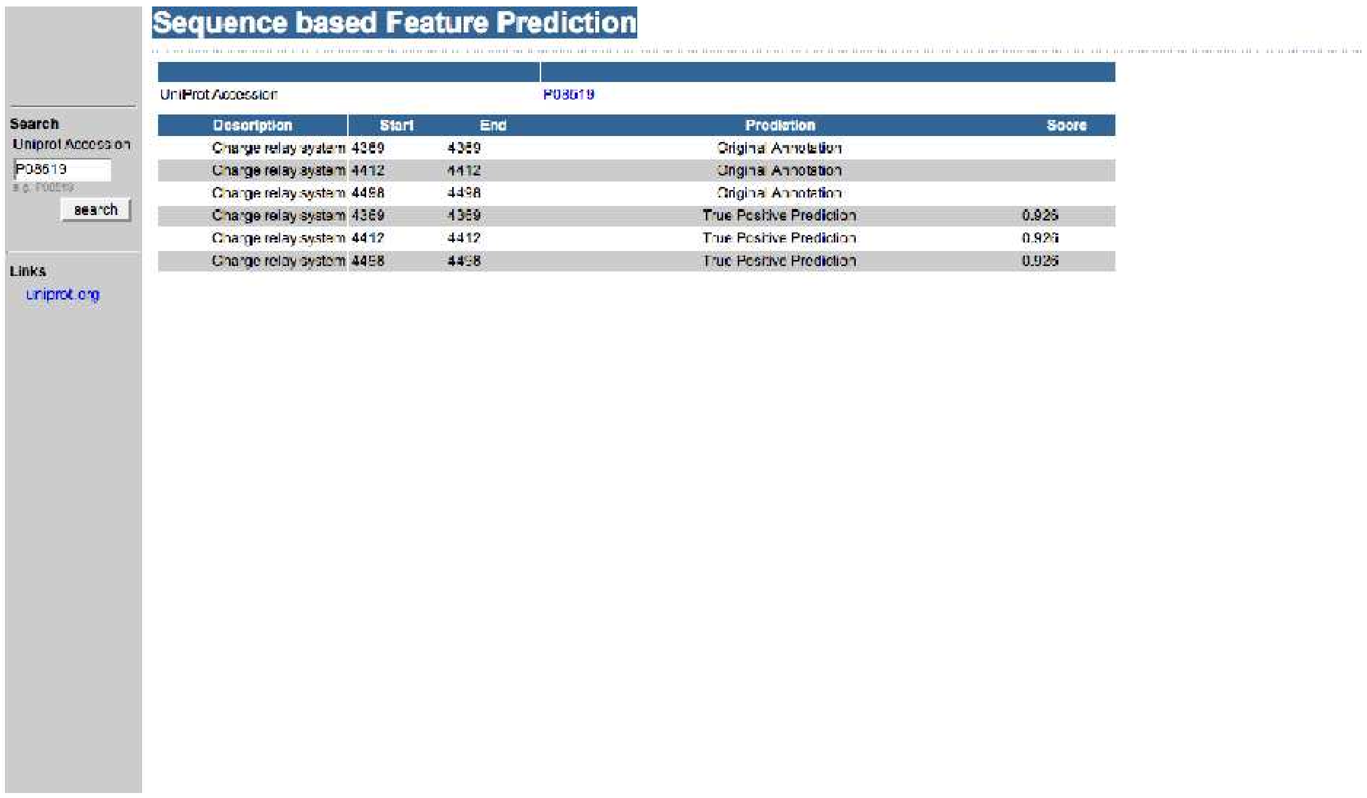
This website allows to browse active site predictions (i.e. descriptions and residues) for Swiss-Prot proteins. The predictions were derived using a combined decision tree and weighted finite state machine approach in a 4-fold cross-validation.

Please paste a UniProt accession number to display manually annotated and predicted active sites ("P08519").

Type	Count	Description
Number of Entries	43378	Number of Proteins in Trainingsset
Number of Active Sites	39056	Number of ActiveSite present in Trainingsset
True Positives Binary Classifier	32008	True Positives Active Site prediction by binary Classifier (Score 0.70)
False Positives Binary Classifier	609	False Positives Active Site prediction by binary Classifier (Score 0.70)
True Positives FSMs	28945	True Positives Active Site prediction by Finite State Machines
False Positives FSMs	918	False Positives Active Site prediction by Finite State Machines

ROC Curve for the Aggregated Algorithm.

Figure 4.6: The webapplication provides a statistical overview over the generated data and shows ROC curves based on the generated data.



Sequence based Feature Prediction

UniProt Accession	P08619

Description	Start	End	Prediction	Score
Charge relay system	4369	4369	Original Annotation	
Charge relay system	4412	4412	Original Annotation	
Charge relay system	4498	4498	Original Annotation	
Charge relay system	4369	4369	True Positive Prediction	0.926
Charge relay system	4412	4412	True Positive Prediction	0.926
Charge relay system	4498	4498	True Positive Prediction	0.926

Figure 4.7: Single UniProt entries can be browsed by querying with an UniProtKB accession. In this case 3 true positive prediction has been made.

5 Results

Two different types of experiments were conducted, which demonstrate the biological relevance and high-throughput capability of the method. The experiments attempted to predict functional site positions, an annotation type that is well maintained in the Swiss-Prot data set and enjoys high biological relevance. For this purchase a two-stage approach was taken, in which the C 4.5 decision tree algorithm is used as a model for predicting the presence of an active site, for example a *proton donor*. This first stage filters out candidate sequences with different activities or no activity at all. Only those sequences which fit this model are used in the second stage to build a finite state machine for the location of the functional site.

5.1 Dataset

The FT (Feature Table) lines provide a precise but simple means for annotation of the sequence data. The table describes regions or sites of interest in the sequence such as which amino acids within the sequence encode the signal or binding sites of the protein as shown in the following example. Active sites are a well maintained feature type with high biological relevance. The Swiss-Prot internal guide for database curators describes the way active sites are annotated:

FT ACT_SITE is used only for enzymes and indicates the residues directly involved in catalysis. There can be 1 or several FT ACT_SITE lines, depending on the nature of the active site. FT ACT_SITE key is used for 1 single amino acid residue. Ideally, active site annotation should be based on a reaction mechanism derived from biochemical and structural studies. Active site residues are usually 100% conserved within a family, however conservative substitutions may occur. Residues that are involved in substrate or cofactor binding but that do not directly participate in catalysis should not be listed *as active site residues. Their role is indicated under FT BINDING/SITE (for single residues) or FT REGION/MOTIF (for a short stretch of residues).*

Both experiments were calculated on the Swiss-Prot section of UniProt version 7.0. 13.7% of all UniProtKB/Swiss-Prot entries were identified as containing active sites. There are 48,527 entries containing an active site in the whole of Swiss-Prot. 44% of the entries have two or more active site annotations. In the experiments only parts of the database were actually used in training sets, which encompassed 43,336 Swiss-Prot proteins having 38,996 annotations in total. The latter number is used for recall calculations. For the decision tree part of the algorithm taxonomic groups, the association to an InterPro family, and the presence of signature hits were used as attributes. Fragments and proteins marked as hypothetical were excluded from training and target sets in both experiments.

5.2 Cross-validation

To show that the two stage algorithm is able to predict the active site in accordance with Swiss-Prot a four-fold cross-validation was carried out against the whole Swiss-Prot data set, where all 43,336 proteins were clustered in InterPro families of between 50 and 500 entries. The cross-validation was used to assess whether a learned algorithm is correct in assigning the predictions to a protein. Cross-validation is a technique in which a dataset divides the known instances into two sets. The first set is used to train the classifier while the second set is held back. The learned classifier is then applied to the second data set and the classification results are compared with the known data. A good prediction system gives good agreement with the known data, therefore, if the predicted classification agrees with the known data from the held-back set one has a true classification. True positives (TP) are those cases where the classifier correctly predicts the occurrence of a property while true negatives (TN) occur when the classifier correctly predicts the lack of a property. When the prediction does not agree with the known data, one has a false classification: Either a false positive (FP) where the classifier predicts a property not known to be present or a false negative (FN) where the classifier predicts the lack of a property known to be present. Different ratios between the FP, FN, TP, and TN are measures of the trustworthness and usefulness of the classifier. Based on these values the true positive rate (recall), false positive rate, and precision of the algorithm can be calculated with the following equations:

$$\text{True Positive Rate} = (\frac{TP}{TP+FN}) \cdot 100\%$$

$$\text{False Positive Rate} = (\frac{TP}{FP+TN}) \cdot 100\%$$

$$\text{Precision} = (\frac{TP}{TP+FP}) \cdot 100\%$$

A good way to visualize the performance of a predictive algorithm is the use of "receiver operation characteristic" (ROC) curves (Beck and Shultz, 1986). These curves have their origin in signal detection and are originally used to distinguish between a positive rate and a false alarm over a noisy channel. The ROC curves plot the number of true positives included in the sample on the vertical axis against the the number of False Positives on the horizontal axis. In the best case the prediction method points straight to the upper left corner of the curve, which means that all positives examples are found without the receiving of any false examples. On the other side, a prediction method with randomly results would give a straight line at an angle of 45 degrees from the horizontal, from bottom left to top right (Witten and Frank, 1999).

5.3 Four-fold cross-validation in Swiss-Prot

The first step of the active site prediction employed a C4.5 decision tree approach to detect proteins with an active site feature annotation. After the decision tree had predicted the presence of an active site with a certain description, the next step was the determination of its location in the sequence using the finite state machine (FSM) approach. The fundamental principle for location prediction was the same for all position-specific feature types, i.e. for active sites, binding sites, substrate regions and metal binding sites. Weighted FSMs can be used to align training and query proteins by calculating the most probable path through these machines. Firstly, each sequence of a positive training instance from the decision tree is represented by a FSM. The composition of all training FSMs results in one probabilistic training model, which represents all the positive training sequences attached to a leaf node in the decision tree. Secondly, a FSM is built, which encodes costs for gaps, weights for specific residues, the location of the site, and mismatch penalties based on the protein subsitution matrix BLOSUM 62 (Henikoff and Henikoff, 1992). Thirdly, the query sequence is represented by a simple unweighted acceptor. The result of this procedure is a weighted FSM, which represents an alignment of the query sequence and the training machine. In this way, functional sites are mapped to amino acids in the query sequence. The described workflow is shown in Figure 5.1.

The performances of the different algorithms were measured separately, calculating the recall precision from the results of the cross-validation. ROC curves as described above have been used for the visualization of the performance. Figure 5.2 shows the performance of the decision tree in a four-fold cross-validation at different threshold scores of the decision tree. A score of 0.70 was found to give the best trade-off between true positive and false positive predictions. At a score of 0.70, the method achieved a recall of 80.3% at a precision of 98.2%. The performance of the location predictor using the weighted FSMs was measured at different threshold scores of the decision tree and is shown in Figure 5.3. The curve shows that the performance of this part of the algorithm has little dependence on the decision tree score. The performance of the aggregated algorithm is shown in Figure 5.4. At a score of 0.70, the method achieved a recall of 72.47% at a precision of 96.9%. The performance of the aggregated algorithm is slighty lower than the performance of the decision tree. This indicates that the recall and the precision of the aggreated algorithm is highly dependent on the performance of the decision tree.

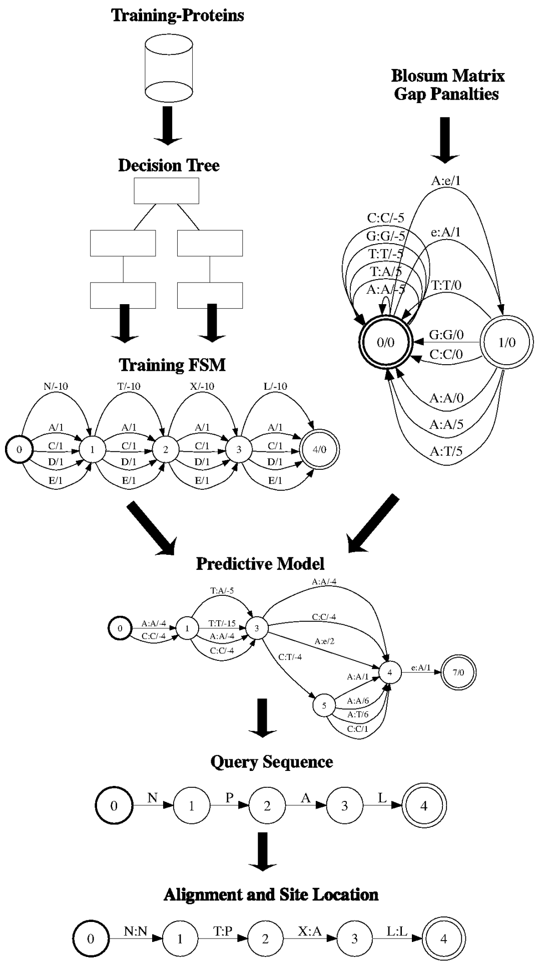


Figure 5.1: Algorithm Flowchart. A decision tree is generated from a set of training proteins. For each node that scores above a minimal threshold, a training FSM is generated. This is composed with an FSM that encodes mismatch penalties, gap penalties and properties of the BLOSUM62 matrix (Henikoff and Henikoff, 1992). The result encodes the predictive model. A further composition with an FSM that represents a query sequence leads to an alignment of the query sequence against the predictive model. The predicted position of the site can be extracted from the resulting FSM.

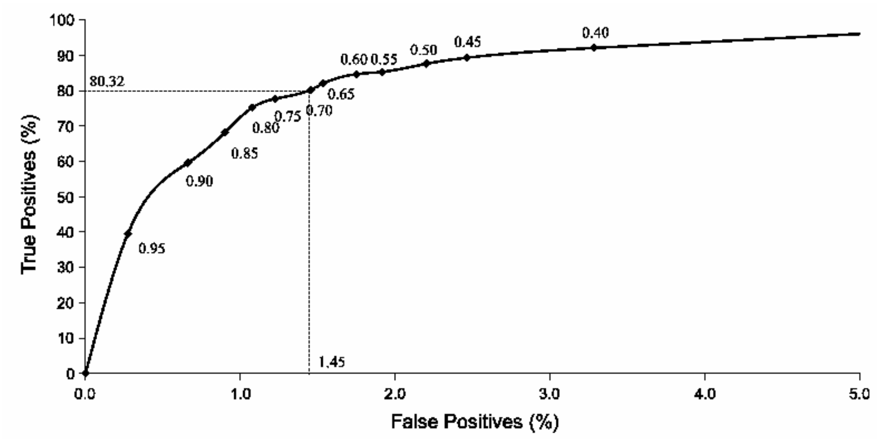


Figure 5.2: ROC Curve for the decision tree. True and false positive rates were calculated for threshold scores between 0.40 and 0.95, below which no prediction is made.

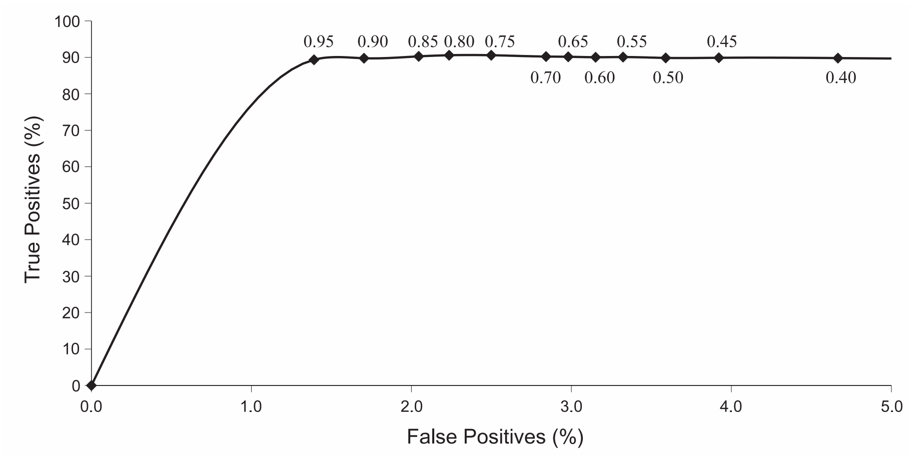


Figure 5.3: ROC curve for location predictor. True and false positive rates were calculated for threshold scores between 0.40 and 0.95, below which no prediction is made.

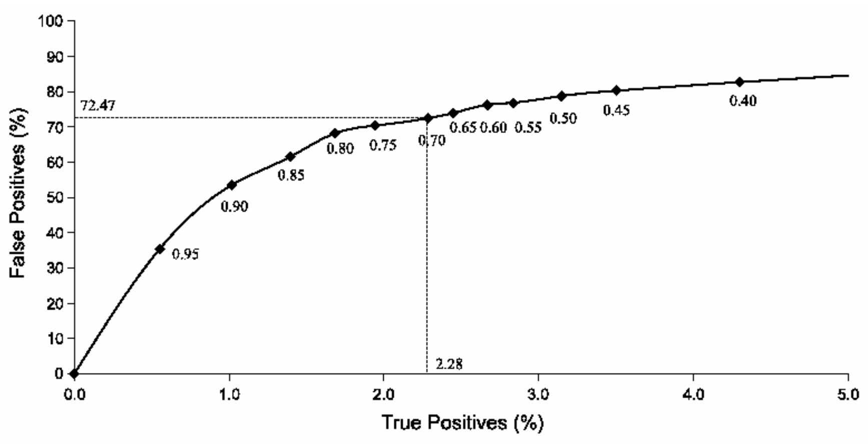


Figure 5.4: ROC curve for the aggregated algorithm. The curve shows that for a value of 0.70, a recall of 72.47% is achieved at a false positive rate of 2.28%.

5.4 Ten-fold cross-validation in the enolase family

The extendibility of the method to other feature types, in particular to predict multiple functional site residues, was tested on proteins belonging to the enolase family. A ten-fold cross-validation was performed for the 286 Swiss-Prot proteins in the training set. Six different target functional sites were present in the training set, these are ACT_SITE (Proton acceptor), ACT_SITE (Proton donor), METAL (Magnesium), REGION (Substrate binding), BINDING (Substrate), BINDING (Substrate; phosphate group). The focus in this experiment was to investigate the performance of the location-finding model. Figure 5.5 shows the precision and the recall achieved by the FSM models for each of the six targets.

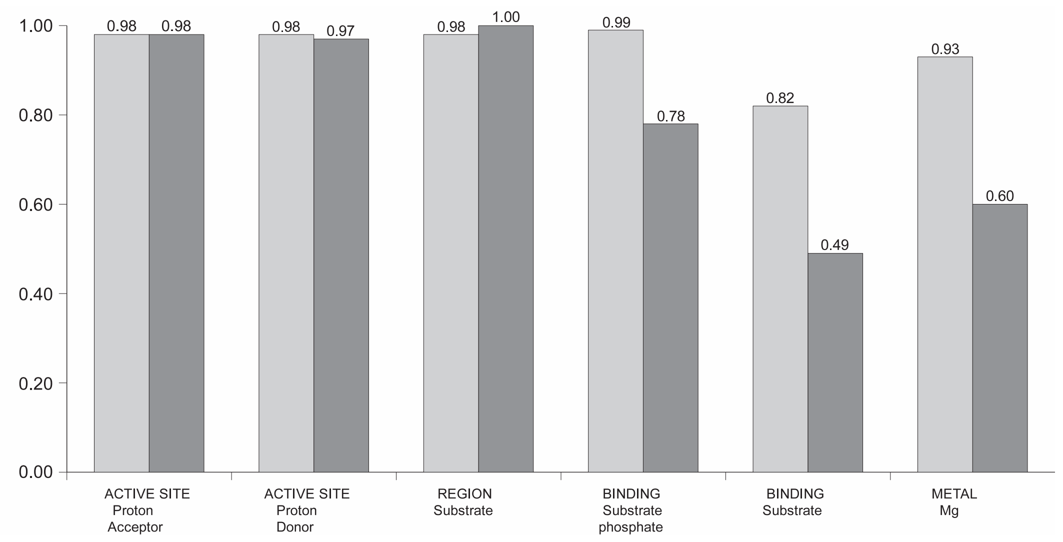


Figure 5.5: Performance of the aggregated algorithm for site specific annotations in enolases. Dark gray columns visualize precisions, light gray columns recall rates.

All of the highly conserved substrate binding regions could be identified at a precision of 99%. The precision for the single substrate binding residues reached 99% for substrate binding in general and 82% for substrate binding at a phosphate group. Recall values were 49% and 78%, respectively. In most database entries, there is one catalytic active magnesium annotation, bound to three residues, i.e. to aspartate, glutamate and aspartate. Precision and recall for this functional site were 93% and 60% respectively.

The algorithm for active sites performed exceptionally well for the enolases. A precision value of 98% was reached for both *Proton Acceptor* and *Proton Donor*, the recall values were 98% and 97% respectively. The results for the enolase analysis show that the Swiss-Prot feature annotations for region and active site keys are easily predictable using the method presented here. In this family, models that predict binding and metal sites perform less effectively. Supervised learning algorithms strongly rely on the consistency of the given training sets, and for active site features, the annotation turned out to be highly consistent in Swiss-Prot. For binding and metal features, the nature of the functional site is different from that of the active sites. However, it was not attempted to model these differences. Thus the same model was used for all single residue sites. An adaptation of the predictive models for binding and for the biology of these sites is likely to improve the performance of the predictor.

5.5 Current state of the system

The system is currently in an evaluation phase but the results are promising. The predictions will be soon integrated in the automated pipeline of UniProtKB. The focus of the analysis was laid on the quality of the generated predictors. Efficiency was not further analyzed, nor was any optimization been conducted to shorten the time required to perform a data-mining run. The time to run a four-fold cross-validation tooks 41 CPU hours. 43378 entries have been analyzed four times, this indicates clearly that the method is not too expensive to be used in a high throughput mode on a regular basis. At the moment the system provides only predictions as a cross-validation on the Swiss-Prot section of the UniprotKB, but the much larger TrEMBL is the focus of automated annotation so this part needs to be integrated.

6 Discussion

6.1 How reliable are the results ?

A cross-validation against the Swiss-Prot section of the UniProt Knowledge base were used to measure the performance of the system. But the presented rates of precision and recall require close examination. Some of the false positives are certainly errors produced by the prediction procedures, but there are also cases where the Swiss-Prot annotation in a few examples seems to be inconsistent in comparison with the rest of the family. In some cases, there may be biological explanations for these exceptions, while in others it may be an annotation error. For example, the decision trees detected 25 proteins in Swiss-Prot with a *biotin / lipoyl attachment* domain (IPR000089) and the PROSITE profile PS50979 (*Biotin carboxylation (BC) domain profile*), all of which had active site annotations. Yet, the location of one of these turned out to be a false positive in the cross validation. In 24 cases the active site position was annotated on an Arginine residue, and on the predicted positon of Q9KWU4 (*Pyruvate carboxylase* in *Bacillus subtilis*), there is also an Arginine. Yet, in the original entry, the active site is annotated on a Lysine residue on a different location. The multiple alignment in Figure 6.1 of all these proteins shows that the sequence is highly conserved across a number of species and that also for Q9KWU4 there is an Arginine residue, where all the other proteins have their active site. The method is useful to detect situations like these, some of which may turn out to be annotation errors after a closer manual examination. The true error rate will therefore be somewhat smaller than the given false positive rates in the results.

```
P05165   RECSIQRRNQKVVEEAPSIFLDAETRRAMGEQAVALARAVKYSSAGTVEFLVDSK--KNFYFLEMNTRLQVEH 333 Human
Q96RQ3   RDCSVQRRHQKIIEEAPAPGIKSEVRKKLGEAAVRAAKAVNYVGAGTVEFIMDSK-HN-FCFMEMNTRLQVEH 344 Human
P11498   RDCSIQRRHQKVVEIAPAAHLDPQLRTRLTSDSVKLAKQVGYENAGTVEFLVDRH--GKHYFIEVNSRLQVEH 333 Human
O00763   RDCSIQRRHQKIVEEAPATIAPLAIFEFMEQCAIRLAKTVGYVSAGTVEYLYS--QDGSFHFLELNPRLQVEH 589 Human
Q05920   RDCSIQRRHQKVVEIAPATHLDPQLRSRLTSDSVKLAKQVGYENAGTVEFLVDKH--GKHYFIEVNSRLQVEH 333 Mouse
Q99MR8   RDCSVQRRHQKIIEEAPAPGINPEVRRKLGEAAVRAAKAVKYVGAGTVEFIMDSR-HN-FYFMEMNTRLQVEH 340 Mouse
Q91ZA3   RECSIQRRNQKVVEEAPSIFLDPETRQAMGEQAVALAKAVKYSSAGTVEFLVDSQ--KNFYFLEMNTRLQVEH 354 Mouse
P11497   RDCSVQRRHQKIIEEAPAAIATPAVFEHMEQCAVKLAKMVGYVSAGTVEYLYS--QDGSFYFLELNPRLQVEH 445 Rat
Q9TTS3   RDCSVQRRHQKIIEEAPAAIATPAVFEHMEQCAVKLARMVGYVSAGTVEYLYS--QDGSFYFLELNPRLQVEH 446 Cow
Q28559   RDCSVQRRHQKIIEEAPAAIATPAVFEHMEQCAVKLARMVGYVSAGTVEYLYS--QDGSFYFLELNPRLQVEH 446 Sheep
P11029   RDCSVQRRHQKIIEEAPASIATSVVFEHMEQCAVKLAKMVGYVSAGTVEYLYS--QDGSFYFLELNPRLQVEH 446 Chicken
Q42777   RDCSVQRRHQKIIEEAPAPNISADFRAQLGVAAVSAAKAVNYYNAGTVEFIVDTV-SDEFYFMEMNTRLQVEH 329 Soybean
P78992   RDCSVQRRHQKVARNCSAKTLPVEVRNAILNDAVKLAKTANYRNAGTAEFLVDSQ--NRHYFIEINPRIQVEH 320 Yeast
Q8X1T3   RDCSVQRRHQKVVEIAPAKTLPVEVRDAILTDAVKLAKAANYRNAGTAEFLVDNQ--NRHYFIEINPRIQVEH 321 Yeast
P78820   RDCSVQRRHQKIIEEAPVTIAPAATFHEMERAAVRLGELVGYASAGTIEYLYEP-ENDRFYFLELNPRLQVEH 398 Yeast
Q42523   RDCSVQRRHQKIIEEAPAPNISEKFRANLGQAAVSAARAVGYYNAGTVEFIVDTE-SDQFYFMEMNTRLQVEH 334 Arabidopsis
Q9HES8   RDCSVQRRHQKVVEIAPAKDLPADVRDRILADAVKLAKSVNYRNAGTAEFLVDQQ--NRYYFIEINPRIQVEH 339 Aspergillus niger
Q9KWU4   RDCSVQRRHQKVIEVAPSVSLSPELRDQICEAAVALAKNVNYINAGTVEFLVAN---NEFYFIEVNPRVQVEH 301 Bacillus subtilis
P24182   LAERDCSMQRRHQKVVEEAPAPGITPELRRYIGERCAKACVDIGYRGAGTFEFLFENGEFYFIEMNTRIQVEH 297 Escheria Coli
P46392   RDCSLQRRFQKLVEEAPAPFLTDAQRKEIHESAKRICKEAHYYGAGTVEYLVGQD--GLISFLEVNTRLQVEH 304 Mycobacterium leprae
P0A508   RECSLQRRHQKVIEEAPSPLLDPQTRERIGVAACNTARCVDYVGAGTVEFIVSAQRPDEFFFMEMNTRLQVEH 299 Mycobacterium leprae
```

Figure 6.1: Alignment of a false positive examples against true positives. The missing active site annotation in Q9KWU4 lies in a highly conserved region.

6.2 Comparison with structure based methods

Most functional site prediction methods rely on structural information. They deliver close models to a narrow range of proteins, an approach that was not taken in this work. These methods measure distances in terms of spatial metric units, which obviously models reality closer than measuring distances in the primary sequence. Yet, in principle, a finite state machine approach is not limited to the primary sequence, but can equally be extended to cover structural information. This approach may lead to higher precision rates but, as a consequence, the efficiency of the predictor may be compromised. Apart from efficiency advantages, the main benefit of this work is its effectiveness to work for a larger number of targets, as primary sequences enjoy a much higher degree of availability than structural data.

6.3 Possible Improvements for the algorithm

The current approach models the region around a known active site from a number of given training sequences. Except for the most trivial decision trees, there are also negative examples available, i.e. sequences which lack the active site feature. It is not infrequent that a catalytic activity is not present in some proteins, although they share large parts of the domain structure with catalytically active samples. There is a good chance that these sequences can be used to assert penalty weights, and that this way the FSMs can be made more specific to the positive examples. Furthermore, in the shown results, all location predictors for active sites were built in the same manner and the performances of the predictors vary for different activities. As a consequence, the models for the weaker predictors could be improved to increase their performance, for example by extending the window size on which they rely.

6.4 Application to TrEMBL proteins

The sheer amount of the available sequence space, in combination with the advances in protein knowledge represented in Swiss-Prot, creates a high efficiency requirement for automated annotation procedures. Meaningful annotation on uncharacterized protein sequences is required so that searches against the complete known sequence space return significant results. As a consequence of the exponential growth, high-throughput methods are required to generate annotation automatically. The results show that the algorithm in this work here is suitable for producing precise, site-specific annotation, in an efficient way. A cross-validation procedure, as presented in the results, gives a good indication of the performance of a particular algorithm. The actual performance for TrEMBL proteins may be different from the values that the cross validation suggest, since the Swiss-Prot section of the UniProtKB is biased towards model organisms and proteins of high scientific interest. A closer examination of how the method affects the state of site-specific feature table annotation of sequences in TrEMBL is currently under way.

7 Conclusion

The goal in this work was to show the potential of using weighted finite state machines (FSMs) in combination with a classification algorithm to predict the presence and location of active sites for any given protein sequence. Other functional sites such as regions, metal and substrate binding sites were accessed with the same fundamental technology and evaluated against the enolase family.

FSMs allow efficient and powerful modelling of biological problems. Simple models such as regular expressions and also more complex ones like hidden Markov models and support vector machines can be built and applied by using weighted FSMs. Thus, they provide a generic concept to model complex sequence data and algorithms using a set of algebraic operations. They deliver a framework with which predictive methods can be combined conveniently to produce more powerful predictors. Structural information, for instance, can equally be represented by an FSM, and this can be composed with the purely sequence-based FSMs.

The AT&T FSM library turns out to be a highly efficient implementation of weighted FSMs, which allows predictions in high-throughput mode. It was shown that with the help of these models more than 70% of all active site features in Swiss-Prot can be recalled at a precision of over 95%. This is a promising footing for further investigations beyond the strictly sequence based methods, beyond fairly simple predicted targets, and beyond the levels of predictive power shown in this work. The results from the ten-fold cross-validation can be browsed at http://www.ebi.ac.uk/interpro/internal-tools/aa/siteprediction.

8 Appendix

8.1 Databases involved

This section gives an overview of most the databases which were used in this thesis. It explains the most important protein databases and pattern databases involved.

8.1.1 Swiss-Prot

Swiss-Prot is a curated database of protein sequences created in 1986 by Amos Bairoch during his PhD and developed by the Swiss Institute of Bioinformatics (SIB) and the European Bioinformatics Institute (EBI). The protein annotation is added by about 50 biologists located in the EBI and the SIB. Each annotation has to pass several quality checks until the final supervision of Amos Bairoch it is checked into the database. In this way a very high level of quality data has been provided throughout the years.

8.1.2 TrEMBL

TrEMBL is an automated translation from GenBank/EMBL/DDBJ DNA sequences to putative or hypothetical protein sequences. Regularly annotation is added both automatically and manually. The quality of the data varies, some entries contain little more than the mere sequence data, others are contain high quality confidential data.

8.1.3 InterPro

The protein signature databases usually categorize similar sets of proteins into the same groups, but at the same time the different methods have their strengths and weaknesses in different areas. Regular expressions have their optimum in short patterns while hidden Markov models are better in finding divergent super-families. InterPro (Biswas et al., 2002) was designed to bring this complementary information under one roof. For every group, there is a small abstract in the database describing the function of the pattern, there is the collection of the proteins belonging to the group and some graphical illustrations for each protein where the pattern in question can be found. Since the ticket for a protein to get into an Interpro entry can be from various sources there are usually many patterns per protein stemming from Prosite, Prints, Pfam, or Prodom.

8.1.4 Prints

The Prints database (Attwood et al., 2003) is a compendium of protein motif fingerprints. A motif is any conserved element of a sequence alignment: it is a local alignment corresponding to a region whose function or structure is known, or its significance may be unknown. It is sufficient that it is conserved, and is hence likely to be predictive of any subsequent occurrence of such a structural/functional region in any other protein sequence. A fingerprint is a set of motifs used to predict the occurrence of similar motifs. A composite or multiple-motif fingerprint contains a number of aligned motifs taken from different parts of a multiple alignment. True family members are then easy to identify by virtue of possessing all elements of the fingerprint, while subfamily members may be identified by possessing only part of it.

8.1.5 Prosite

Prosite (Hulo et al., 2006) is a database of protein families and domains. It is based on the observation that, while there is a huge number of different proteins, most of them can be grouped, on the basis of similarities in their sequences, into a limited number of families. This database is actually divided into two sections. The greater part contains regular expressions for protein patterns. A big disadvantage is that they do not allow variable gaps where arbitrary acids can be present. An alternative approach to motif based pattern recognition is to distil the sequence information within complete scoring tables, or profiles.

8.1.6 Pfam

Pfam (Bateman et al., 2004) is a large collection of multiple sequence alignments and hidden Markov models covering many common protein domains and families. Each family in Pfam is represented by two multiple sequence alignments and two profile-hidden Markov model. Pfam has two large series of functionally uncharacterized families, known as domains of unknown function and uncharacterized protein families. Tracking the number of these functions and families gives some idea of how many families in Pfam are uncharacterized and how this number has changed over time.

8.2 Swiss-Prot datatypes

The following datatypes are used in Swiss-Prot:

8.2.1 ID

The ID line of each entry contains the entry name, the data class (`STANDARD` = Swiss-Prot, `PRELIMINARY` = TrEMBL), the molecule type (`PRT` = protein) and the length of the sequence:

```
ID   GRAA_HUMAN     STANDARD;      PRT;   262 AA.
```

8.2.2 DE

Each entry has exactly one description line which summarizes the most important information about the protein:

```
DE   GRANZYME A PRECURSOR (EC 3.4.21.78) (CYTOTOXIC T-LYMPHOCYTE PROTEINASE
DE   1) (HANUKKAH FACTOR) (H FACTOR) (HF) (GRANZYME 1) (CTL TRYPTASE)
DE   (FRAGMENTIN 1).
```

8.2.3 OS

The OS (Organism Species) line specifies the organism(s) which was the source of the stored sequence:

```
OS   Mus musculus (Mouse), Rattus norvegicus (Rat).
```

8.2.4 RN, RP, RC, RX, RA, RT, RL

These lines collect the information about the citations which indicate the sources from which the data has been abstracted:

```
RP   SEQUENCE OF 29-53.
RX   MEDLINE; 88330824.
RA   Poe M., Bennett C.D., Biddison W.E., Blake J.T., Norton G.P.,
RA   Rodkey J.A., Sigal N.H., Turner R.V., Wu J.K., Zweerink H.J.;
RT   "Human cytotoxic lymphocyte tryptase. Its purification from granules
RT   and the characterization of inhibitor and substrate specificity.";
RL   J. Biol. Chem. 263:13215-13222(1988).
```

8.2.5 CC

The CC lines provide textual comments on the entry, and may be used to convey any useful information. The comment blocks are arranged according to what is designated as 'topics' which allow a mapping of a comment into a category:

```
CC   -!- DEVELOPMENTAL STAGE: EXPRESSED IN EMBRYONIC AND EARLY LARVAL
CC       STAGES.
```

8.2.6 DR

The DR (Database cross-Reference) lines are used to store information related to Swiss-Prot entries. Here, references to signature hit database such as InterPro, Prosite, Pfam, and others are stored:

```
DR   INTERPRO; IPR001254; -.
DR   PROSITE; PS00968; ANTENNA_COMP_ALPHA; 1.
```

8.2.7 KW

The KW (KeyWord) lines provide information that can be used to generate indexes of the sequence entries based on functional, structural, or other categories.

```
KW   Oxidoreductase; Acetylation.
```

8.2.8 FT

The FT (Feature Table) lines provide a precise but simple means for the annotation of the sequence data. The table describes regions or sites of interest in the sequence such as which amino acids out of the sequence encode the signal or binding sites (cf. Appendix A) of the protein.

```
FT   SIGNAL        1     26
FT   ACT_SITE     69     69       CHARGE RELAY SYSTEM (BY SIMILARITY).
FT   ACT_SITE    114    114       CHARGE RELAY SYSTEM (BY SIMILARITY).
```

8.2.9 SQ

The SQ line contains the length of the sequence in amino acids ('AA') followed by the molecular weight ('MW') rounded to the nearest mass unit (Dalton) and the sequence 64-bit CRC (Cyclic Redundancy Check) value ('CRC64'). The algorithm to compute the CRC64 is described in the ISO 3309 standard. It should be noted that, while in theory, two different sequences could have the same CRC64 value, the likelihood that this would happen is quite low. This line is followed by the core sequence data:

```
SQ   SEQUENCE   233 AA;  25630 MW;  146A1B48A1475C86 CRC64;
     GDVEKGKKIF IMKCSQCHTV EKGGKHKTGP NLHGLFGRKT GQAPGYSYTA ANKNKGIIWG
     EDTLMEYLEN PKKYIPGTKM IFVGIKKKEE RADLIAYLKK ATNE
```

9 Bibliography

Attwood, T. K., Bradley, P., Flower, D. R., Gaulton, A., Maudling, N., Mitchell, A. L., Moulton, G., Nordle, A., Paine, K., Taylor, P., Uddin, A., and Zygouri, C. (2003). PRINTS and its automatic supplement, prePRINTS. *Nucleic Acids Res*, 31(1):400–402.

Bateman, A., Coin, L., Durbin, R., Finn, R., Hollich, V., Griffiths-Jones, S., Khanna, A., Marshall, M., Moxon, S., Sonnhammer, E., Studholme, D., Yeats, C., and Eddy, S. (2004). The Pfam protein families database. *Nucleic Acids Res*, 32(Database issue):D138–D141.

Beck, J. and Shultz, E. (1986). The use of relative operating characteristic (roc) curves in test performance evaluation. *Arch Pathol Lab Med.*, 110(1):13–20.

Biswas, M., O'Rourke, J., Camon, E., Fraser, G., Kanapin, A., Karavidopoulou, Y., Kersey, P., Kriventseva, E., Mittard, V., Mulder, N., Phan, I., Servant, F., and Apweiler, R. (2002). Applications of InterPro in protein annotation and genome analysis. *Brief Bioinform*, 3(3):285–295.

Cavaness, C. (2004). *Programming Jakarta Struts.* O'Reilly.

Cortes, C., Haffner, P., and Mohri, M. (2004). Rational kernels: Theory and algorithms. *The Journal of Machine Learning Research*, 5(1):1035–1062.

Eddy, S. (1998). Profile hidden markov models. *Bioinformatics*, 14(3):755–763.

Gamma, E. (1997). *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison-Wesley.

Goslin, J. (1988). *The Java Language Specification.* Sun.

Henikoff, S. and Henikoff, J. (1992). Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. USA*, 89(5):10915 – 10919.

Hopcroft, J. and Ullman, J. (1979). *Introduction to Automata Theory, Languages and Computation.* Addison -Wesley.

Hulo, N., Bairoch, A., Bulliard, V., Cerutti, L., Castro, E. D., Langendijk-Genevaux, P. S., Pagni, M., and Sigrist, C. J. A. (2006). The PROSITE database. *Nucleic Acids Res*, 34(Database issue):D227–D230.

Hyafil, L. (1976). Constructing optimal binary decision trees is np-complete, inform. *Inform. Process*, 5(4):15–17.

Kernighan, B. W. and Ritchie, D. M. (1988). *The C programming Language.* Prentice-Hall, Inc.

Kretschmann, E., Fleischmann, W., and Apweiler, R. (2001). Automatic rule generation for protein annotation with the C4.5 data mining algorithm applied on SWISS-PROT. *Bioinformatics*, 17(10):920–926.

Letunic, I., Copley, R., Pils, B., Pinkert, S., Schultz, J., and Bork, P. (2006). SMART 5: domains in the context of genomes and networks. *Nucleic Acids Res*, 34(Database issue):D257–D260.

Mohri, M. (1997). Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311.

Quevillon, E., Silventoinen, V., Pillai, S., Harte, N., Mulder, N., Apweiler, R., and Lopez, R. (2005). InterProScan: protein domains identifier. *Nucleic Acids Res*, 34(Database issue):D227–D230.

Quinlan, J. R. (1988). *C 4.5 Programs for Machine Learning.* Prentice-Hall, Inc.

Rabiner, L. and Juang, B. (1986). An introduction to hidden markov models. *ASSP Magazine, IEEE*, 1(1):4 – 16.

Witten, I. H. and Frank, E. (1999). *Datamining Practical Machine Learning Tools and Techniques with Java Implementions.* Prentice-Hall, Inc.

Wu, C., Apweiler, R., Bairoch, A., Natale, D., Barker, W., Boeckmann, B., Ferro, S., Gasteiger, E., Huang, H., Lopez, R., Magrane, M., Martin, M., Mazumder, R., O'Donovan, C., Redaschi, N., and Suzek, B. (2006). The Universal Protein Resource (UniProt): an expanding universe of protein information. *Nucleic Acids Res*, 34(Database issue):D187–D191.