Fachhochschule Weihenstephan
University of Applied Sciences

Diplomarbeit

Automated Annotation: Filtering Erroneous Protein Data

Author: Daniela Wieser

Date: June 2004

Supervisors:

Ernst Kretschmann	Prof. Dr. Bernhard Haubold
EMBL Outstation - Hinxton	Fachhochschule Weihenstephan
European Bioinformatics Institute	Fachbereich Biotechnologie
Wellcome Trust Genome Campus	Bioinformatikzentrum, E.10
Cambridge, CB10 1SD	85350 Freising
United Kingdom	Germany
phone:+44 (0)1223 / 492585	phone: +49 (0)8161 / 71-5274

Chapter 1

Acknowledgements

First of all I want to thank Ernst Kretschmann who was the supervisor of my master thesis at the EBI. He did an excellent job while supervising me and he contributed to the project with a lot of good ideas and suggestions. I also want to thank my supervisor from the university Prof. Dr. Bernhard Haubold for his help concerning any kind of questions and Prof. Dr. Frank Lesske for proof-reading my master thesis. Thank you very much to Rolf Apweiler, the group leader of the EBI Sequence Database Group, for supporting student projects and hence for giving me the possibility to work on the Xanthippe project. Rather than to thank each person individually, I want to mention that I would surely have failed my thesis without the help of the friendly and supportive staff, especially the members of the Automated Annotation group at the EBI. Special thanks go to my boyfriend Markus Brosch for letting me go to Cambridge without any misgivings and for encouraging me whenever I felt lost.

Eidesstattliche Erklaerung

Gemaess §23 Abs. 6 der Pruefungsordnung

Ich erklaere hiermit an Eides statt, dass die vorliegende Arbeit von mir selbst und ohne fremde Hilfe verfasst und noch nicht anderweitig fuer Pruefungszwecke vorgelegt wurde.

Es wurden keine als die angegebenen Quellen oder Hilfsmittel benutzt. Woertliche und sinngemaesse Zitate sind als solche gekennzeichneet.

Cambridge, 10. Juni 2004

Daniela Wieser

Contents

List of Figures

List of Tables

Listings

Chapter 2

Introduction

Automated Annotation in UniProt

The protein databases Swiss-Prot, TrEMBL and the PIR have been unified into a single resource under the UniProt effort [Apweiler *et al.*, 2004] since November 2003. From the Swiss-Prot section of UniProt, users obtain a manually curated data-set of high quality. Annotation is produced by a literature curation process and concerns mainly protein descriptions, i.e. names and synonyms, comments, keywords and sequence features. Large parts of the TrEMBL section, the non-curated remainder of UniProt, provide few, if any, of the above-mentioned value-adding annotation items. Now we are in the age of high-throughput sequencing and the manual curation process has not been able to cope with the avalanche of newly available sequence data and as a consequence the proportion of well-annotated protein data is constantly shrinking. Since this situation is unsatisfactory, various applications to generate annotation automatically have been proposed and implemented in recent years (see, for example, [Prlic *et al.*, 2004]; [Fleischmann *et al.*, 1999]).

One major aspect of the UniProt effort is to establish an automated annotation pipeline to provide users with predicted annotation, especially for otherwise little- or non-annotated database entries. The predictive annotation rule sets generated in the RuleBase [Biswas *et al.*, 2001] and Spearmint [Kretschmann *et al.*, 2001] projects are executed on a regular basis. The results of these approaches, which increase the information content of UniProt considerably, are presented to the scientific community on the project's web pages (http://www.ebi.uniprot.org). They are shown as prescriptive annotation on a separate layer and suggest annotation without any modifications of the original data itself. Different colors in the HTML view make obvious which annotation item is generated automatically and at which level of confidence they are produced.

Listing 2.1 shows the abridged UniProt protein entry 'Q8UF43'. The entry holds some information about the protein like its ID, the species, organism classification and the sequence. Beside this core data which is known and present for each UniProt entry, there is a short description "*Alcohol dehydrogenase*" and the keyword "*Complete Proteome*". The latter annotation data item indicates that the protein is expressed by an organism for which the genome has been sequenced completely. This information does not concern the protein itself, but rather the organism from which it was extracted and hence this keyword is not very meaningful concerning the function of the protein. The only information about the protein function which a scientist gets from this entry is reduced to a short description that indicates which enzyme the protein is ("*Alcohol dehydrogenase*").

Listing 2.1: Abridged UniProt protein entry. Beside the core data and a short description only one keyword was assigned to it.

```
ID  Q8UF43
DE  Alcohol dehydrogenase.
GN  ADH or ATU1557 or AGR_C_2867.
OS  Agrobacterium tumefaciens (strain C58 / ATCC 33970).
OC  Bacteria; Proteobacteria; Alphaproteobact.; Rhizobiales;
OC  Rhizobiaceae; Rhizobium/Agrobacterium group; Agrobact..
OX  NCBI_TaxID=176299;
KW  Complete proteome.
SQ  SEQUENCE 368 AA; 39154 MW; 7BA714CF2AD464BE CRC64;
    MFDASITIRG GTTMFTTSAY ACDDGSSPMK LATIRRRDPG ...
```

Listing 2.2 illustrates the same protein entry enriched with predicted annotation. Predicted annotation is labelled with two asterisks at the beginning of a line. Lines that start without an asterisk belong to the original entry. The prediction system Spearmint suggests that this enzyme is classified as *EC 1.1.1.1* (see line 5). The confidence value of this prediction lies at 90.3% (see line 6). Comments were predicted from both RuleBase and Spearmint (see line 11 to 20), as well as keywords (see line 20 to 28).

Listing 2.2: Protein function prediction in an UniProt entry (abridged). Two asterisks at the beginning of a line indicate that the following information was derived from one of the prediction systems Spearmint (SM) or RuleBase (RB).

```
 1  ID Q8UF43
 2  DE Alcohol dehydrogenase{EI2}.
 3  **DE  < predicted: >
 4  **DE  < enzyme classification: >
 5  **DE  EC 1.1.1.1{SM0}
 6  **PY  SM0: SPM028292 (C90.3);
 7  OS Agrobacterium tumefaciens(strain C58/ATCC 33970).
 8  OC Bacteria; Proteobact.; Alphaproteobact.;
 9  OC Rhizobiales;Rhizobiaceae;
10  OC Rhizobium/Agrobact. group; Agrobact..
11  **CC  < predicted: >
12  **CC  -!- CATALYTIC ACTIVITY: An alcohol + NAD(+) =
13  **CC   an aldehyde or ketone +NADH{SM1}.
14  **CC  -!- COFACTOR: Zinc{RB2}.
15  **CC  -!- COFACTOR: Binds 2 zinc ions per subunit{SM3}.
16  **CC  -!- SIMILARITY: Belongs to the zinc-containing
17  **CC      alcohol dehydrogenase family{RB4,SM3}.
18  **PY  SM3: SPM004003 (C91.1);
19  **PY  RB2: RU000467 1.0(98.57);
20  **PY  RB4: RU000468 1.0(94.87);
21  KW Complete proteome{EP6}.
22  **KW  < predicted: >
23  **KW  Metal-binding{RB2}
24  **KW  Oxidoreductase{RB2,SM3}
25  **KW  Zinc{RB2,SM3}
26  **KW  NAD{SM10}
27  **PY  SM3: SPM004003 (C91.1);
28  **PY  RB2: RU000467 1.0(98.57);
```

Both listings show abridged UniProt entries that have been extracted from the EBI UniProt web pages where they can be viewed as complete entries. After searching for the protein entry with the ID ”Q8UF43” and clicking on the result set, the listing 2.1 can be found on the basic UniProt view. Listing 2.2 also contains predicted annotation which can be viewed after clicking on the extended UniProt view (not yet in the flat file but in the HTML pages). The listings illustrate that although there is not much information provided in the original protein entry, scientists can obtain more information about the protein once the automated annotation data has been published.

Automated Annotation - Difficulties

A cross-validation of predictive models against Swiss-Prot and various surveys of the produced data have shown that some of the automatically generated annotation is erroneous. The fact that both predictive systems applied in the automated annotation pipeline rely on protein families, domains and sequence signatures is one source of these errors. The InterPro database [Mulder *et al.*, 2003] provides this data by assigning a protein sequence to a particular domain or family based on the presence of a single signature hit. Whenever false positive hits are encountered, data-mining applications have to deal with erroneous input data. A cross-reference to an InterPro group is not removed, even though the signature hit which lead to the assignment of the protein to this group is eventually found to be false positive. It is a non-trivial task to render each and every annotation rule robust against this possibility. False positives appear over a wide range, with some hitting to related families or remotely similar biochemical properties, and some even occurring as entirely random events. Another source of errors lies in the bias between training and target sets, which are the Swiss-Prot and TrEMBL sections of UniProt, respectively. Some situations in the target set are not represented in the training set at all and can therefore not be resolved by mining algorithms using examples in the training set only. For instance, an annotation rule that was exported in Spearmint added the keyword ”*Nuclear protein*” to all entries in the TrEMBL section of UniProt having the InterPro domain IPR001005 (*”Myb DNA-binding domain”*) and the SMART [Schultz *et al.*, 2000] hit SM00717 (*”SANT*

SWI3, ADA2, N-CoR and TFIIIB" DNA-binding domains"). The keyword is annotated in all of the 70 non-hypothetical Swiss-Prot proteins containing the InterPro domain and the SMART hit. In the target set however, protein Q819P5 (*"Prespore specific transcriptional activator rsfA"*) fulfils the conditions of this annotation rule, despite its belonging to the kingdom of bacteria. There were no bacterial proteins in the training set and so the algorithm was not trained using a fully representative set of instances. Since there is no way of knowing what proteins are going to be present in a future TrEMBL database version, full representation of all circumstances in the training set can never be achieved. Yet, a close analysis of the output of the annotation rule immediately leads to a straightforward method for filtering out this particular erroneous annotation. Bacteria do not possess nuclear proteins because of their lack of a nucleus. In all cases the "*Nuclear protein*" keyword annotated on bacterial proteins is wrong, regardless of the origin of the annotation, which could be predictive systems, data imports or even human curation. This can be expressed as a simple exclusion rule, which when applied to the TrEMBL section of UniProt not only removes 66 wrong keyword predictions produced by automated annotation, but also spots the same error in some imports (e.g. in the bacterial protein Q93HH7).

Listing 2.3 illustrates an example of a wrong annotation. The protein entry with the accession number Q93HH7 that is currently part of the UniProt TrEMBL section was extracted from a bacteria (see OC line 5). Despite this information, the keyword "*Nuclear protein*" was assigned to this protein entry (see line 8). This import is obviously wrong and should be either removed or at least be marked as possibly wrong to avoid the propagation of those errors to other data sources and to prevent users from processing this information.

Listing 2.3: TrEMBL entry Q93HH7 (abridged). Note that the keyword ”*Nuclear Protein*” should not be in line 8, as the described protein does belong do the bacteria, which do not possess a nucleus.

```
1  AC Q93HH7;
2  DE HoxX-like protein{EI1}.
3  GN SAV7365{EP3}.
4  OS Streptomyces avermitilis.
5  OC Bacteria; Actinobacteria; Actinobacteridae;
6  OC Actinomycetales;Streptomycineae; Streptomycetaceae;
7  OC Streptomyces.
8  KW Homeobox{EP5}; Nuclear protein{EP5};
9  KW Complete proteome{EP3}.
```

It is widely believed that human intelligence is necessary to verify any information that is assigned to a protein and hence that only human experts are able to frame reliable exclusion rules. They are able to combine the biological knowledge that they have gathered over years and can look up missing information in publications, books or databases to decide on the correctness of an assigned annotation.

It is true, that with respect to a given single protein human experts still have advantages compared to computational methods as scientist can look up any information in a huge amount of data sources that are useful to analyze a particular protein. However, there are other criteria that might support computational methods. Firstly, biologists do not have the time to scan each protein entry and cross-reference them with a large number of other data sources to check if they are correctly annotated. The current TrEMBL Release (07-Jun-2004) contains 1 062 416 entries (http://www.ebi.ac.uk/trembl/). It would take thousands of human database curators to effectively check more than one million entries in the TrEMBL database. Secondly, even well-educated human curators are often not able to combine all information that is known about proteins because they cannot survey the overwhelming amount of protein data available. In many cases, the data simply can not be processed in the human brain. Lacking such abilities humans often cannot spot the erroneous data.

Aim of the master thesis

The aim of this master thesis was to develop computational methods that overcome the mentioned difficulties by using data-mining approaches. The general idea of data-mining is to look for hidden patterns in a large quantity of data and to find out which patterns are connected or not connected to others, whereas a connection can either lead to the inference or to the exclusion of a certain event. As an example of data-mining let us look for all bacterial proteins and for all proteins with the annotated keyword ”*Chloroplast*” in a protein database. After assembling all protein entries containing these patterns, a data-mining process tries to connect them and infers the absence of the keyword ”*Cholorplast*” for all bacterial proteins, i.e. the event of annotating a bacterial protein with that keyword should never happen. An exclusion rule for this situation can be created.

The project was divided into the following three steps:
1. Algorithm development
2. Comparison of two algorithms - Simple Mapping Mechanism and Decision Trees
3. Development of a comprehensive data-mining environment

In the following chapter the principles of the two approaches are explained. Afterwards the in Java implemented system is presented, which is designed to automatically mine for exclusion rules to a far deeper level than in the obvious example of the keyword *Nuclear protein* in bacteria, and to apply these rules to predicted, imported and literature-curated annotations in UniProt database entries. Finally, the results section illustrates the benefit of Xanthippe concerning the quality of the UniProt database, whereas the discussion will point out that Xanthippe also can contribute considerably to improve the data quantity in the UniProt database. The results of the application are shown alongside the automated annotation part of the UniProt entry as prescriptive annotation (http://www.ebi.uniprot.org). The project was named ”Xanthippe” after Socrates’ renowned shrewish wife, due to the nature of the system to scrutinize the output of other systems and, if required, mark it as questionable.

Chapter 3

Systems and Methods

3.1 Introduction - Choosing a suitable training-set

The task of this master thesis was to develop computational methods to detect erroneous protein annotation in the TrEMBL database by using data-mining approaches. Before a mining process can be started, a database is needed on which the data-mining training process can be performed. The intention to detect erroneous protein annotation in the TrEMBL database, suggests to use the Swiss-Prot database as a reference. Swiss-Prot currently contains over 150 000 well-annotated protein entries from which the mining algorithm can extract data (as per Swiss-Prot Release 43.5 of 07-Jun-2004; http://us.expasy.org/sprot/). Furthermore, the Swiss-Prot format and the TrEMBL format are basically identical and they use similar patterns for the description of the proteins. Hence, it is possible to extract exclusion rules from the Swiss-Prot database and to apply them to entries in TrEMBL. However, there is also information in the Swiss-Prot entries that cannot be used for creating the exclusion rules, i.e. information that is annotated in the Swiss-Prot entries, but not in the TrEMBL entries. If rules were created with this data, they would not apply to most of the TrEMBL entries. It is reasonable to use only those entities for the condition of the rules that are present or pre-calculated for each protein entry in UniProt, i.e. in each Swiss-Prot and in each TrEMBL entry. These are data such as the taxonomy of the organism, from which the protein was extracted, or the result of InterProScan [Zdobnov *et al.*, 2001], which automatically classifies sequences into families and domains and detects hits to signature databases. This kind of information is considered to be core data and is available for each protein in UniProt.

It turns out that the presence and absence of annotation items is in many cases a function of the distribution of core data in the protein entry. The cases where annotation items are implied are mined by predictive systems such as RuleBase and Spearmint. It is evident that there are also cases where annotation items are excluded by core data, but the predictors do not use this concept.

In the introductory example, the absence of ”*Nuclear protein*” keyword annotation in bacterial proteins was discussed, a fact that was deduced by using biological reasoning. In the following, two methods are presented which produce this and similar exclusion rules using a statistical data-mining approach. Since April 2004 they are integrated in the EBI automated annotation pipeline and used as a system to avoid annotation errors in UniProt entries. Annotation items are marked as potentially wrong whenever the core data distribution in the entry suggests that the item should be absent.

3.2 Method 1: Simple Mapping Mechanism

The given example exploits the fact that organisms of the kingdom of bacteria do not possess any nuclear proteins. This simple implication can be detected automatically by examining the distributions of taxa (core data) and ”*Nuclear protein*” keywords (annotation) in the Swiss-Prot section of UniProt. Out of 153 017 entries (as per Swiss-Prot Release 43.5 of 07-Jun-2004; http://us.expasy.org/sprot/), there are 65 465 bacterial and 9454 nuclear proteins. Assuming a random distribution of these data items, an overlap would be expected, i.e. bacterial proteins with ”*Nuclear protein*” annotation. The expected value is 4046 instances, while the observed overlap is 0. In a database where these data items are statistically distributed this would be an extremely unlikely situation with a likelihood of less than $4.6 \cdot 10^{-3485}$. This value is so small that not only can the assumption of a random distribution be discarded but there is also a good indication that the two entities are mutually exclusive.

Reconstruction of the example:
The example was assorted after browsing the UniProt web pages (http://www.ebi.uniprot.org) on 08-June-2004. If the same queries are posed on a later date the result sets are expected to be larger, as the Swiss-Prot database is constantly growing (see figure 3.1).

- Go to http://www.ebi.uniprot.org
- Determine the number of bacterial proteins in Swiss-Prot.
  Therefore query the UniProt database as follows:
  1. Select *Advanced Text Search*
  2. Select a library: *Swiss-Prot*
  3. Query line type: *Organism Classification (OC)*
  4. Enter the query text: *Bacteria*
  5. Result: ***65 465 entries***
- Determine the number of ”*Nuclear protein*” keywords.
  Therefore query the UniProt database as follows:
  1. Select *Advanced Text Search*
  2. Select a library: *Swiss-Prot*
  3. Query line type: *Keywords*
  4. Enter the query text: *Nuclear protein*
  5. Result: ***9454 entries***
- Determine the expected number of bacterial proteins in a random distributed Swiss-Prot database with the keyword *Nuclear protein*.
  1. Determine percentage of bacterial proteins in Swiss-Prot.
     (65 465 * 100%) /153 017 = 42.8%

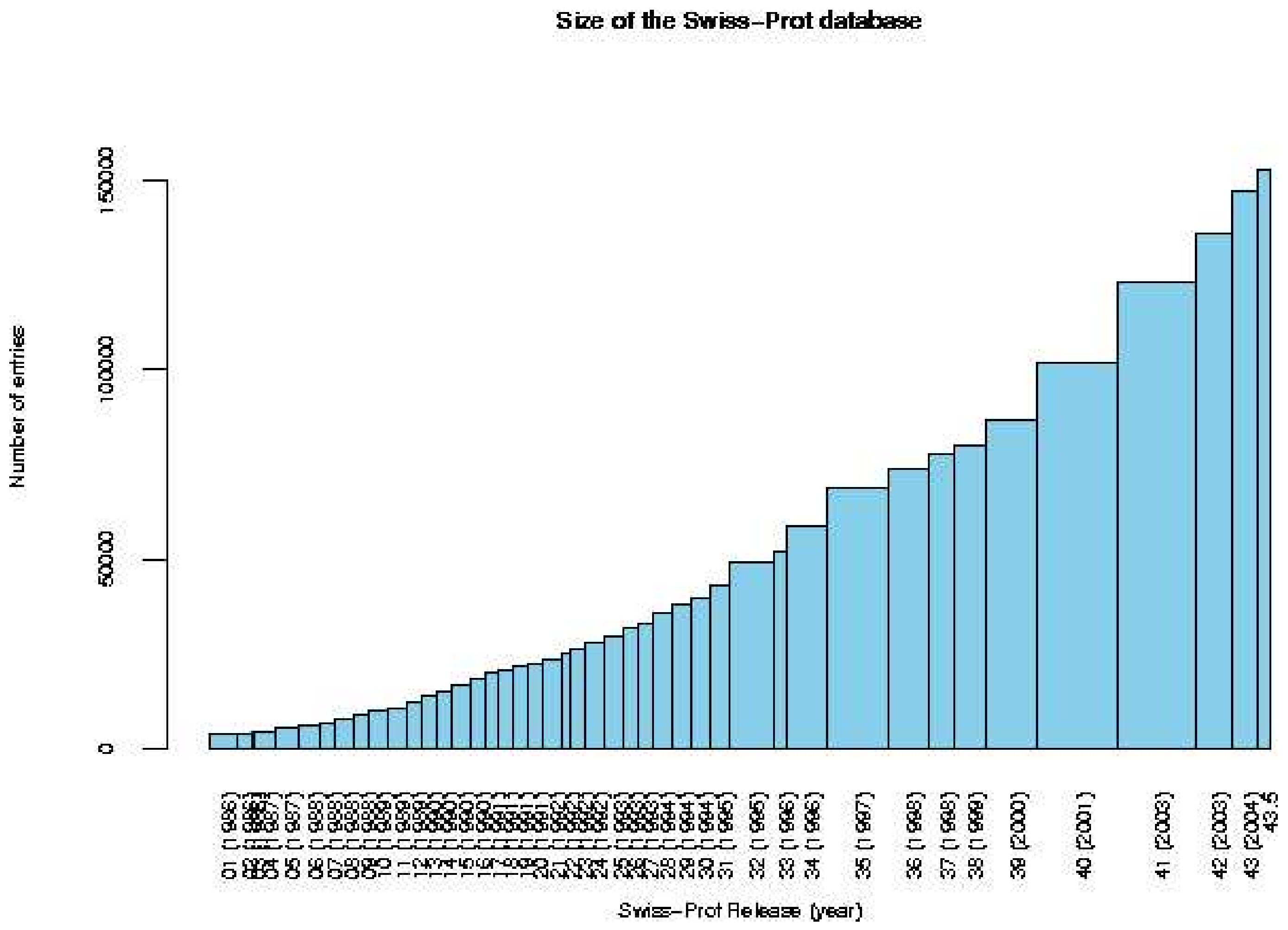

Fig. 3.1: Size of the Swiss-Prot database. See http://ca.expasy.org/sprot/relnotes/relstat.html.

2. Determine the number of bacterial proteins with the keyword *Nuclear protein*. 0.428*9454 = **4046**

- Determine the observed number of bacterial proteins with the keyword *Nuclear protein* in the Swiss-Prot database.

  1. Select: Data Set Manager (top of the page)
  2. Select a set A: *oc bacteria*
  3. Select a set B: *kw nuclear protein*
  4. Select a set operator: *Icon: Intersection*
  5. Click Button *Combine Data Sets*
  6. Result: *The combination of data sets has resulted in no proteins being found. Please retry.*

- Estimate the likelihood of observing no proteins in a random distributed database. Therefore compute the probability for each keyword ”*Nuclear protein*” (total number: 9454) to belong to a non-bacterial protein (= 57.2% of the database) :

  $0.572^{9454} = \mathbf{2.6 \cdot 10^{-2294}}$

  Note: This computation is an estimation of the probability. It gives the upper threshold. The actual probability is even lower, as the number of non-bacterial protein to which a nuclear protein can be assigned becomes smaller after each multiplication.

It is a simple task to design an algorithm that iterates through each taxon-keyword combination not present in Swiss-Prot and calculates a value for the probability of not observing an overlap. A threshold can be determined empirically, above which the combination is exported as an exclusion rule and can be applied to data in the TrEMBL section. At a threshold value of $1 \cdot 10^{-10}$ around 4000 such exclusions are generated.

Obviously, further mappings from core data can be used to contradict an annotation for the corresponding proteins. Signature hits of the protein sequence and InterPro families or domains are particularly interesting and could be exploited to exclude annotation items. Yet, there are drawbacks to this approach. Proteins from a family or having a hit to a specific signature belong to their group according to specific properties. Unlike the set of bacterial proteins that covers a wide range of functions and hence contains a large number of distinct annotation items, the set of proteins belonging to a protein family by comparison comprises a very limited range. In every protein family or group of proteins hitting a common sequence signature, most annotations are absent, and only a few are present. Literally millions of rules can be created and the execution of these rules is far too inefficient in terms of application time.

Another drawback is that these exclusion rules will supposedly not detect a large proportion of the actual annotation errors. It is hardly likely that a high-quality prediction mechanism or the literature curation process produces many annotations entirely non-specific to the protein families to which a given target protein belongs. There is a better chance that annotation items which are specific to the family of a target protein are affected by prediction errors. Unfortunately, such errors cannot be detected by using simple exclusions. This algorithm is designed to contradict only non-specific combinations, i.e. those which never occur in the given protein families. A better way of targeting them is to use a decision tree algorithm very similar to that employed in generating the Spearmint rule set.

3.3 Method 2: Decision Tree Algorithm

While the mapping approach groups proteins globally into those having a core property and those not having it, exclusion trees are generated from a local and comparatively small set of training entities (about 50 proteins). The training sets are chosen to contain proteins that are reasonably similar to each other, for instance all Swiss-Prot entries belonging to a given InterPro family or domain. Because of the similarity between the proteins, the annotation in such groups is usually limited to a fairly small number of annotation items. Most errors will affect these items rather than those not occurring inside this group. To find a contradictor to prevent such errors, a decision tree for each annotation that occurs in a given training set is produced using the C4.5 algorithm. C4.5 is an implementation of a decision tree learning algorithm called ID3 [Quinlan, 1993]. It produces a tree of classificatory decisions with respect to a previously chosen target classification (e.g. learns from all proteins belonging to IPR000001). The binary decision tree which is produced consists of nodes and leaves. The nodes represent the conditions (e.g. IPR000001) of an exclusion rule whereas the leaves hold annotation data items (e.g. keyword *Aids*). The leaves of the trees are examined to find the negative instances, i.e. those that do not have a particular annotation, and rules are then derived, which describe the absence of the annotation. The set of all generated absence rules eventually serves as a contradictive system.

The following example illustrates the exclusion tree approach. Mining for the ”*Mitochondrion*” keyword in InterPro domain IPR009056 (”*Cytochrome c*”) produces the decision tree shown in figure 3.2.

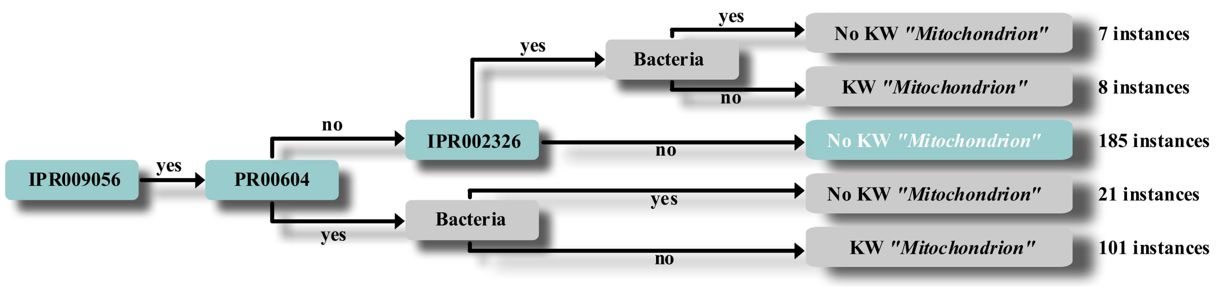

Fig. 3.2: Example Decision Tree.

The tree is entirely generated on a statistical basis but it reflects some basic biological facts. In any case the prediction of the keyword is excluded from the kingdom of bacteria, who do not have mitochondria. More interesting for an exclusion rule generator are those proteins that neither hit to PRINTS [Attwood *et al.*, 2003] PR00604 (”*Cytochrome c, class IA/ IB*”) nor belong to InterPro family IPR002326 (”*Cytochrome c1*”). Both families are found in mitochondria, photosynthetic bacteria and other prokaryotes. The remainder of the InterPro domain IPR009056 (”*Cy-*

tochrome c"), according to the decision tree, are not localized in the mitochondria.

185 instances in Swiss-Prot belong to InterPro IPR009056 and do not hit PRINTS PR00604 or InterPro IPR002326 (last node with blue background in figure 3.2) and none of them has the *"Mitochondrion"* annotation. For there to be a *"Mitochondrion"* keyword predicted in this group, the protein would have to have at least one of the hits PR00604 or IPR002326; otherwise it will be contradicted by Xanthippe.

For the Swiss-Prot protein YIPP_DROME (*"Yippee protein"*) from *Drosophila melanogaster*, the Spearmint system predicts a *"Mitochondrion"* keyword. It uses a decision tree generated from training proteins in IPR000345 (*"Cytochrome c heme-binding site"*). This protein belongs to IPR009056 but hits neither PR00604 nor IPR002326, and hence, the keyword *"Mitochondrion"* is not annotated in the original entry. Xanthippe exclusion trees detect this error made by Spearmint and mark it as possibly erroneous.

Chapter 4

Software Architecture

4.1 Introduction

The previous chapter describes how a simple mapping mechanism and a decision tree algorithm could be used to create exclusion rules that aim to detect erroneous protein data. After a preliminary implementation of these two methods, it was soon found that both are able to detect a significant number of annotation errors.

Due to the sound results of this test application, it was clear that the algorithms have to be integrated in the automated annotation pipeline, so that the prediction systems RuleBase and Spearmint can benefit from the Xanthippe exclusion rules.

However, with the decision to integrate Xanthippe into the automated annotation pipeline, the requirements of the application changed. Integration into the pipeline requires that Xanthippe performs a production run every two weeks to establish the exclusion rules, which can then be applied to the actual protein entries every two weeks, i.e. whenever a new UniProt release is launched. In addition, there is a need to continue with improvements of Xanthippe in experimental runs to find optimal parameter settings, to test a new mining algorithm or to check the results based on different training-sets. A re-implementation of the test version was advisable, which served the purpose to try out the general performance of the algorithms, but which was not designed to integrate both production and experimental runs. The software model suggested after the decision to integrate Xanthippe in the automated annotation pipeline is introduced in the next section.

4.2 Model

A software model for the data-mining environment was designed that allows both production and experimental runs to be performed within the same Xanthippe application. Figure 4.1 shows the general principle of this model.

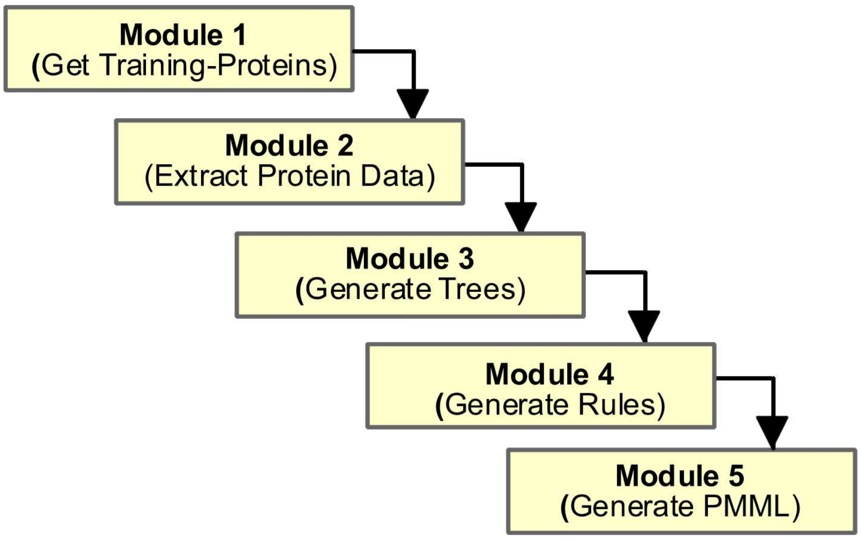

Fig. 4.1: Waterfall model demonstrating the modularity of the application. In this example, the application consists of five modules with different tasks. The notes in brackets give the task of the corresponding module.

The structure is similar to a waterfall model, which is known to proceed linearly through the phases of a software development process, or in our case through different modules of the Xanthippe application. The application is divided into several subunits, which build modules with defined tasks. Example modules include composing of the training-set, tree generating or rule generating. Detailed information about each module is given in the implementation section and in the appendix. Each of the components is regarded as a mandatory step in the program flow. The modules are executed in a certain order, i.e. some modules have to be processed before others. For instance, the step in which the training-set is loaded is always executed before rule generation.

Even though the modules have a defined task, several implementations of a module can exist, as there might be different ways of fulfilling that task. Module 1 in figure 4.1 can obtain the Entry Set either from the local disk or from an external database. These are two different ways of executing the task that require two different implementations of module 1. The software model is designed to allow an easy exchange of the different module implementations. Before running the application, the user can decide which implementation of the training-set loading module is required for a particular mining process. Defined interfaces have to be implemented to make sure that the subsequent module always matches the previous one, regardless of which implementation is used in the mining process. For example, an interface defines that module 1 has a method that returns the protein Entry Set and that module 2 always provides a method that takes the protein Entry Set as a parameter. Hence it is always guaranteed that module 1 can transfer its output to module 2.

It is also possible to return to a previous step after the execution of a given step. This may be necessary if a chosen module implementation does not lead to the desired results. If Xanthippe notices at run-time, that the training-set does not yield enough information to allow data-mining, the application can return to the first step and load another training-set. Thus, the program can be refined during the execution of a run. Figure 4.2 illustrates the concept of returning to a previous step and switching the module implementation.

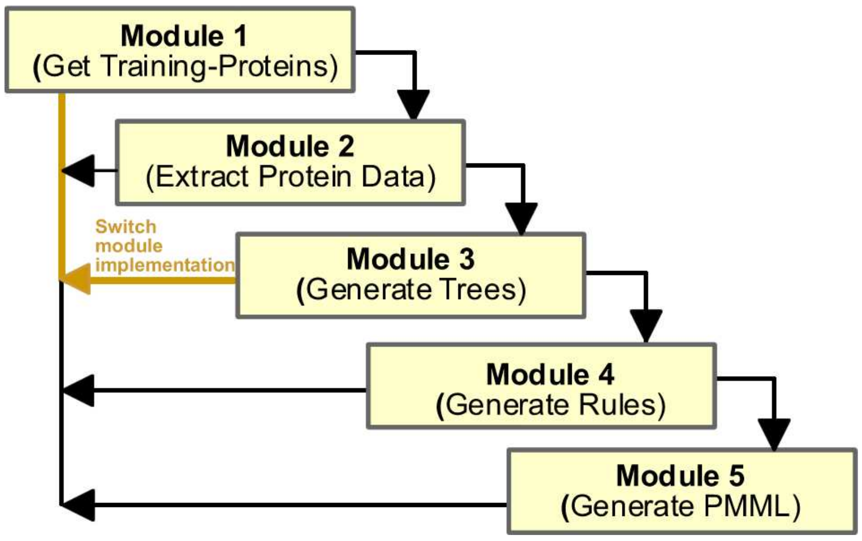

Fig. 4.2: Waterfall model that demonstrates the exchangeability of the application. After executing module 3 (tree generator) the application returns to module 1 to switch the entry set and to start again with the data-mining process.

4.3 Benefits of the model

The software architecture, which is similar to a waterfall model, offers the following benefits to Xanthippe.

4.3.1 Extensibility

The Xanthippe application is easy to extend and users can upgrade the program according to their needs. This feature arises from the modularity of the application. An abstract class can be used to predetermine the behaviour of each of the encapsulated modules.

Two different possibilities are available to extend the application. Firstly, existing module implementations can be extended as long as the guidelines of the abstract class are followed and thus functionality can be added. Secondly, custom implementations of the modules can be written, that can either be exchanged with default implementations or inserted between existing steps.

For instance, it is reasonable to remove redundancy in the set of generated rules as it can take hours to apply all of the rules and in addition it is unnecessary to apply the same rule several times. One possibility to integrate this functionality in the data-mining routine is to assign this task to module 4 (Figure 4.1). Also, a separate module could be inserted between modules 4 and 5, which is responsible for removing redundancies before PMML generation starts. PMML generation is usually the last step of the data-mining application. PMML stands for Predictive Model Markup Language. The exclusion rules are translated into this language which is understood by the database, so that the rules can finally be applied.

4.3.2 Flexibility

Another benefit of the new software model is its great flexibility, arising from the configurability of the program and again from its modularity.

The module implementations and parameter settings with which Xanthippe performs the run can be specified in a configuration file. The program requests the values of the parameters during the run and thus the implementing classes of the modules are plugged into the data-mining routine according to the specification in the configuration file. This leads to a flexible application where the user can choose between several provided or custom module implementations according to the users needs, and where parameters are easy to set. The program flow can be easily changed by simply choosing different module implementations and adapting the configuration file. Another advantage of a configuration file is that the user is aware of all important parameters that influence the program. It allows the user to experiment by changing these parameters in only one file rather than searching and adapting them in the program code.

A short example of a configuration file is given in listing 4.1 in which two different items are specified. Firstly, the mining targets are defined. In this example the mining process would search for all keywords starting with letters A (starting letter specified in line 2), B, C, D or H (ending letter in line 2) except for the keyword *"Albinism"* (line 3). Next, preprocessor parameters are defined. A module implementation to be plugged into the data-mining routine is defined on line 5, followed by parameters for the preprocessing method on line 6. It is mandatory to specify an implementing class for each module (not shown in this example); the advantage of this is that by configuring all important parameters and modules the user always knows how the program works.

Listing 4.1: Short Example configuration file

```
1 # Mining Targets
2 keywordInclude = A, H
3 keywordExclude = Albinism, Albinism
4
5 # Preprocessing Module Specification
6 preprocessor = uk.ac.ebi.seqdbg.datamining.preprocessing.PreProcessor
7 preprocessorParameters = removeFragments, removeHypotheticals
```

Splitting the program into modules means that, if new ideas are tested, only the corresponding modules have to be re-implemented whereas the remaining program part is not affected at all. The modules can be arbitrarily exchanged giving the user flexibility in choosing which modules to use. The user can choose between several module implementations and determine which of them he wants to use for a run. In a later run he can swap module implementations via configuration file. For instance, several implementations of the module that loads the training-set are possible. For one run the implementation of module 1 can be used that loads the trainings-set from the hard disk and for a second run training sets can be loaded from an external database.

4.3.3 Cross-validation Option

The software model allows cross-validation to be performed. The rules of a production run are applied to the whole TrEMBL database and are tested every two weeks, i.e. when the production run of the automated annotation system is performed. It is not advisable to apply the rules of an experimental run in which new approaches or parameter settings are tested to the whole TrEMBL database. Firstly there is a need to check the rules more often than every two weeks and secondly an application of the testing rules to the whole database would take too long. Although only a full application to the whole TrEMBL database provides exact results about the efficiency of the rules, an application to a small target set that is similar to the training set is usually sufficient to get a rough idea of the quality of the rules. It is more likely that a rule is triggered if it is applied to a protein entry that belongs to the same family as the proteins which belong to the training-set because the protein already fulfils some preconditions of the rule. For example, if all proteins in the training-set were extracted from a bacterium, it is reasonable to apply the generated rules only to bacterial proteins. Instead of applying the rules to the whole TrEMBL database, a small target-set can be used on which a cross-validation of the created rules is performed.

A cross-validation can easily be implemented and plugged into the proposed software model. A new module can be implemented or an existing module can be exchanged in which the entry-set is separated into a training and a target-set. For instance, the entry set in the module ”Get Entry-Set” can be split into two sets. E.g. 90% of the entry-set build the training set and are used to create exclusion rules whereas the target set consists of the remaining 10% of the entry-set on which the rules are applied for testing reasons.

These two sets can be used for cross-validation in which the quality of the rules can be examined. Particular investigations on some rules or special annotation errors can be performed without starting the whole production run, but rather by applying the rules to the small target-set. If several runs are performed with different parameter settings, Xanthippe will create a varying set of exclusion rules. Each of these sets will probably detect different errors or a distinct amount of errors if their rules are applied to the target set, because they were created according to varying conditions. One can check which of the sets detects most errors, and one can identify those parameter settings for which the application creates the best rules.

4.3.4 Production run vs. Experimental run

The software model is designed in such a way, that both production runs and experimental runs can be executed with the same application. Production runs are performed every two weeks in order to establish those rules that are applied to the current UniProt release. The results of the production run can be viewed on the UniProt pages. The experimental runs aim to test new approaches or new parameter settings in order to improve the system. Their results are not published, but only studied internally. It is a challenging task to integrate both types of runs in one software system: frequent implementation changes due to experiments require a lot of effort and can put the production run at risk. For example, the current mining process is optimized for the C4.5 decision tree algorithm. It uses a lot of classes and methods from the WEKA package (data-mining software written in Java [Witten *et al.*,2000] for tree operations, e.g. for composing the input format required by decision tree algorithm, for construction of the tree itself or for post-processing the tree. These tree operations are embedded in several classes of the Xanthippe application. Consequently modifications of many classes would be necessary if the performance of a different tree algorithm with a different data-mining software, e.g. CART [Breiman *et al.*, 1984], were to be checked. It cannot be guaranteed that any changes are successfully implemented and tested within two weeks and thus there is a danger that the program can not be executed for the production run, if it is not possible to switch back to the old implementation. The software model allows production runs and experiments to be performed with the same application without compromising the production run. Two configuration files are provided, one for experimental runs and one for production runs. Those modules and parameter settings which are stable and already demonstrated to perform well can be specified in the configuration file for the production run, whereas the experimental configuration file serves as the playground in which arbitrary module implementations and parameter settings can be configured. The suitable configuration is passed to the software application before execution of the program. Reconstructing the program flow of the production run every two weeks is not a problem, because the user only has to swap the configuration files.

4.3.5 Common Data Mining Environment

Even though Xanthippe and Spearmint have similar approaches they were originally developed independently. It is reasonable to combine the two systems to avoid duplicate maintenance effort, as many steps in the program flow of the two systems are similar or even identical.

For example, the fact that all proteins with the status 'hypothetical' should be removed from the training-set applies to both systems, since hypothetical proteins are classified as insufficiently annotated and are inappropriate for any rule finding process. The mining process itself is another example of the similarity of Spearmint and Xanthippe. Both systems scan a training-set of well-annotated Swiss-Prot proteins and return a complete decision tree. Apart from minor differences in parameter settings the same general flow is processed. In both cases the returned tree provides complete information for creating exclusion or annotation rules. Hence, it is usually unnecessary to calculate a separate decision tree for each of the systems.

The software model as described here provides the foundation for a common data-mining environment, in which Spearmint and Xanthippe can run in parallel in order to avoid duplicate program code. The modules and their assigned task in figure 4.1 apply equally to the generation of exclusion rules as well as to the generation of annotation rules. Some module implementations can be used for both systems, such as loading the entry set. Others are adapted to the corresponding system; for example, the implementation of the module that generates rules is different for Spearmint and Xanthippe. Again, the configuration file specifies one system and associated modules to execute. It is also possible to specify both systems in the configuration file. If the systems run concurrently they can perform their task more efficiently, because some program parts only have to be calculated once for both programs, such as the generation of decision trees.

4.4 Implementation

The waterfall model described in the previous section introduced the general concept of the software model on which the data-mining application is based. This section describes the implementation details of the model. There is a need to refine the model introduced in the previous section before it is implemented in order to encapsulate additional program parts. A final software model proposes the implementation of eleven modules which are described below.

1. **Loading of the Entry Set**
   A set of well-annotated proteins is needed at the beginning of a run, as the data-mining process needs to scan all these proteins in order to extract useful information that will be used to create either annotation or exclusion rules. The ”EntrySet” (B.1) is the datastructure that holds this set of well-annotated proteins. As data-mining is not possible without these proteins each program cycle starts by loading an ”EntrySet” on which the data-mining can be driven. If there are several ”EntrySets” that are supposed to be used for the mining process the application will be processed in a recursive manner. For example, a loader class selects the first ”EntrySet” and when the mining process is finished, a different mining process is started with the next set.

The loading step acts as the data-mining application's entry point and is therefore mandatory in order to perform the data-mining. A Java class is used to compose, manage and pass a set of training-proteins to the application. As there are differing ways of performing the afore mentionend tasks, several classes are possible to implements the tasks. A common abstract class defines the core operations of this mining-step, which has to be implemented from all other classes assigned to the package the package "uk.ac.ebi.seqdbg.datamining.loading". One loading class has to be specified in the configuration file which is passed to the mining application. An error message will turn up, if no loading step is configured. Further information on the implemented classes that belong to this module are given in the appendix C.1

2. **Pre-Processing**

   Before the actual data-mining can begin a pre-processor has to prepare the loaded training-set. Protein entries that are not suitable for the mining process should be removed from the training-set and others may need to be modified to make them appropriate.

   For example, fragments should be removed. The signature hits that come with a fragment are found after aligning the fragment to signature databases like PRINTS or Prosite. The fragment is a part of a long sequence, and presumably more signature hits would be found if the whole sequence was aligned to the databases. Thus a lot of information is missing if data-mining is performed on fragments,because additional signatures of their complete sequences are disregarded. Hence there would be a risk that non-specific rules were created. A common abstract class defines the core operations of this mining-step, which has to be implemented from all other classes assigned to this package. One class performing the pre-processing step has to be specified in the configuration file which is passed to the mining application. An error message will turn up, if no pre-processing step is configured. The implementation details of the classes of the package "datamining.preprocessing" are described in appendix C.2

3. **Training-Set Splitting**

   For cross-validation reasons, the splitting step divides the training-set into two smaller ones. The first set can serve as a training-set, i.e. rules are created after mining this set. The second set holds those proteins on which the rules are applied to check their performance. We expect, that none of the exclusion rules should trigger if applied to the target proteins. However, should they trigger it suggests that the rule finding process was not thourough. Appendix C.3 gives information about the classes that belong to this module.

4. **Arff-Generating**

   The protein data in the training-set has to be available in a suitable format that allows the application to perform data-mining operations on it's member proteins. The classes in the package Arff convert the data into an appropriate format. Firstly they must retrieve the data from the database, via a seperate package, then the data can be converted into the ARFF format (described below). The ARFF file generation is divided into two steps in this data-mining application. Firstly, the ARFF file is prepared, i.e. the signatures of the training-proteins are collected and written in an incomplete file. Secondly, this prepared ARFF file is completed in adding an annotation data item,whereas the application requires an own ARFF file for each annotation data item. The way to prepare or rather to generate this ARFF is user-definable and managed in the Package 'Arff', its classes are described in C.4.

   An **ARFF** (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. ARFF files were developed by the Machine Learning Project at the Department of Computer Science of The University of Waikato for use with the Weka machine learning software [Witten *et al.*,2000]. The Weka software is open source data-mining software written in Java.

   The Xanthippe data-mining application uses an ARFF file to store information about the proteins which belong to the training-set. An example is given

Attributes→ **Instances↓**	SM00020	PS50070	SM00473	IPR000001	Polymorphism
HGFA_HUMAN	+	+	-	+	-
HGFA_MOUSE	+	+	-	+	-
HGFL_HUMAN	+	+	+	+	+
HGFL_MOUSE	+	+	+	+	-

Table 4.1: Example ARFF-File

in table 4.1. The training-set consists of four haemoglobin proteins which all belong to the InterPro cluster IPR000001. A class from the WEKA package is used to find out which attributes are shared from these proteins and creates the shown table. In each row the table contains a protein instance and in each column the protein attributes for the instances are indicated with ”+” and ”-” signs. For instance, the protein HGFL_HUMAN does belong to the protein families SM00020, PS50070, SM00473, IPR000001 and it has the keyword ”*Polymorphism*” annotated.

The ARFF file contains all the information used for the mining process. Once it is generated it is passed to the next module that is responsible for the tree generation.

5. **Data Mining**
   The actual data-mining takes place in the mining step. An Arff object, which holds the information extracted from the training-set is first passed into the routine. This object is then searched by a mining algorithm, which ultimatly returns a decision tree, that suggests an annotation data item for a protein entry, if it fulfils certain conditions.
   (See appendix C.5 for classes in the datamining package).

6. **Post-Processing**
   The classes in the post-processing packages aim to adapt the trees which were

returned by the mining step. Differing decision tree algorithms, when used in the mining step will return different trees. For example, if the DefaultDataMiner (see C.5) was used, the decision tree will have the format that is defined in the Weka package (open source data-mining software). This tree might not be completely appropriate for the following program flow, but still can be readily adapted in the post-processing step. (See C.6 for implementation details).

7. **Selecting**

   By the time the mining process reaches the selection step, a lot of decision trees that hold either exclusion or annotation rules, are available. So far no tree has been rejected, regardless of which rules it consists of. Some of the trees are not well suited for the given input proteins. For example, a rule may have been added to a tree even though there were some proteins in the training-set that behaved exceptionally to it. However, this mining step aims to exclude all trees that are found to be not sufficiently reliable enough to be applied to the target protein set.(See C.7 for implementation details).

8. **Crossvalidation**

   The cross-validation step is the last component of the actual mining cycle. In this step it is possible to check the performance of the annotation rules directly after they have been created, filtered but before they have been applied. If the application of the rules deliver good test results, the mining process stops at this place. Otherwise, the mining process can be redefined and the parameter settings be changed so that eventually the best possible rules are extracted from the training-set. (See C.8 for implementation details).

9. **Exporting**

   The mining process itself is finished when the program arrives at the exporting step. The ”Exporting” module intends to save the final rules that were generated in the mining process. (See C.9 for implementation details).

10. **RedundancyRemover***

    Some of the rules that were generated might be redundant. An additional module ”Redundancy Remover” was implemented that identifies and removes these redundancies. This step finally shortens the runtime of the application process of the Xanthippe rules, as less rules need to be applied.

11. **PMML Generating***

    The rules have to be converted to a certain format, the PMML(Predictive Model Markup Language). This is a XML based mark up language to describe statistical and data-mining models [Grossman *et al.*, 1999]. The PMML allows to apply the rules on UniProt entries. This module intends to generate these PMML files.

All presented modules are part of the data-mining application and they are mentionend in the order in which they should be executed. Some of them are exchangeable, i.e. they can be adapted according to different problems. There are also fixed components which should not be exchanged. Class implementations are provided for them, which are recommended to be used without any changes. Sometimes these fixed components concern complete modules and in other cases it concerns data structures that belong either to an exchangable or fixed module. The fixed modules are indicated in the above mentioned list with an asterisk at the end (modules 10 and 11). Both types of components are introduced in the following. Firstly, the implementation of the fixed components is addressed.

4.4.1 Fixed Components

While some program parts have to be adapted to special conditions from run to run, others are fixed. The implementations of fixed parts are not supposed to be changed or adapted by the user. They are provided as completed modules (see modules 10 and 11) because they are always needed in the same form, regardless of how the remaining application is implemented.

Some data structures in the exchangeable modules are also fixed, as they are essential to the application. Essential parts of the program are necessary to fulfil the fundamental task of the application, which is to perform data-mining on a set of proteins and to return a set of annotation rules. A set of proteins is an example for an essential program part, as it is needed several times in each mining process. For instance, it is composed in module 1, pre-processed in module 2 and split in module 3. It is reasonable to have a data structure for the set of proteins that can be used whenever operations on this set are performed. After analyzing the behaviour of the training-set a data structure 'EntrySet' was implemented. It contains fields and access methods useful for working with the training-set. There are operations for adding an entry to the training-set or operations that can be used to save information concerning the origin of the training set (see Java-doc in the attached CD-ROM).

An example for a fixed module is PMML generation (module 11). The PMML file holds the exclusion rules which are translated into a markup language, and is used for the application of the rules on the UniProt database. The PMML file has a specified structure that should be followed in order to guarantee an error-free application. The module which is responsible for the PMML generating is therefore fixed and should not be changed. It requests a flatfile with the generated rules and translates it into PMML.

Additional fixed data structures have been implemented, the most important ones are present in the appendix, where also further implementation details can be viewed (see attached CD-ROM for full implementation).

4.4.2 Exchangeable Components

Even though some components are unchangeable, there are sections which can be adapted to special requirements (module 1 - module 9). Separate class implementations for each of those modules are reasonable, because they simplify the module exchange process. A favoured class implementation is specified in the configuration file, which is plugged into the data-mining routine when the program starts.

An abstract class for each of the modules is defined, that guarantees the integration of the implementations in the program flow. This abstract class defines an interface that has to be followed if a concrete sub-class is implemented. The abstract handler classes are assigned to several packages, which also provide one or more implementing subclasses of the corresponding superclass. A class that implements the module's task has to extend the abstract class and it has to implement its provided method declarations. The implementation details are up to the class, but it has to follow the given guidelines.

For instance there is a package that is responsible for the loading of the training-set (module 1). An abstract class ”TrainingSetLoader” provides methods that have to be implemented from subclasses. Two subclasses of the ”TrainingSetLoader” were implemented in the final Xanthippe application. One subclass loads the training-set from the local disk (”LocalLoader”) and another subclass loads it from an external database (”InterProLoader”). The module implementation that is supposed to be used in the run, is specified in the configuration file. If a different implementation of a module is required for the next run, a different sub-class is specified. The task and the implementation details of the exchangeable components are described in the appendix (see attached CD-ROM for full implementation).

Chapter 5

Results

Since two inherently different sets of exclusion rules were generated, the results are presented separately to allow a thorough analysis of the individual systems.

Organism to Keyword Exclusions

This system is rather stable in its output, it being unlikely that in the future organisms will be found that contradict the known fundamental properties of their taxonomical backgrounds. Since each data mining application on reasonably sized data sets produces false positive predictions, we chose to present the once exported exclusion rules to a biological expert who manually picked out those of biological value. One artefact that could be detected in this procedure was the apparent absence of ATP-binding proteins in a range of venomous snakes. The statistical approach produced a value far below the threshold and exported an exclusion between the taxonomy of these snakes and the *"ATP-binding"* keyword. In reality this exclusion does not denote any exceptional metabolic properties of these animals, but more the high level of scientific interest in protein samples of their venom. None of these bind ATP, while the rest of the proteome is not represented in the Swiss-Prot section of UniProt.

In the following the results of applying 700 curated rules are discussed. They were applied only when there was no example of that particular combination in Swiss-Prot. They were considered to be biological facts, so their confidence value was set to 100%.

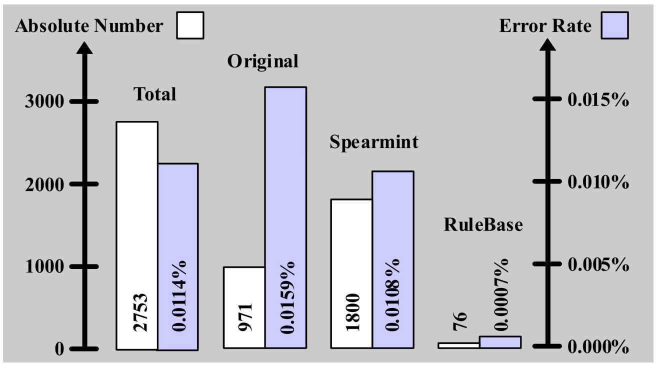

Fig. 5.1: Number of cases found in the TrEMBL section of UniProt, which were contradicted by Xanthippe exclusions from organism to keyword (white). Note, that the individual numbers do not add up, since overlaps occur. The blue part shows the error rate, i.e. how many contradictions where found compared to the amount of annotations provided by the individual sections.

A cross-validation to Swiss-Prot is therefore unnecessary and only their performance in the TrEMBL section of UniProt and on automated annotation is given in figure 5.1.

Spearmint was found to produce the largest amount of annotation errors in total numbers, while the original annotation on the entries, usually imports from EMBL [Kulikova *et al.*, 2004], had the highest error rate.

Exclusion Trees

This method is highly sensitive to minor changes in the distribution of core data in the training set, and hence exclusion trees are exported with every release of a new version of Swiss-Prot. In general the new exports differ from former results. They frequently query for related signature hits or on different levels in the taxonomy tree, but if applied they usually produce the same contradictions. Therefore, a curation step as in the first method is not feasible and the artefacts produced by this method have to be accepted and investigated.

Exclusion trees were generated for three inherently different annotation items: key-

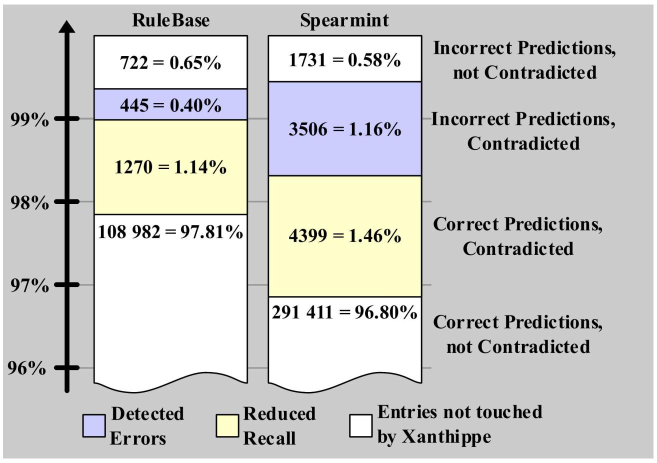

Fig. 5.2: Performance of Xanthippe exclusion trees on keyword predictions from Rule-Base and Spearmint.

words, protein names and comments. Keywords consist of a controlled vocabulary of approximately 850 distinct words that are highly accessible to data mining algorithms. Protein names are less controlled and comments can be used for entirely free text annotation. Closer examination reveals that large parts of the latter annotation items in fact also consist of controlled vocabulary. They are kept consistent in the Swiss-Prot section of UniProt, occur in multiple entries and can therefore be picked by algorithms working on a statistical basis.

The results are given individually for rules from Spearmint and RuleBase to show that the method performs well on predictive methods from entirely different backgrounds. RuleBase is an expert-curated annotation system, where the rules are created on a basis of biological reason and only partly on statistical considerations. Spearmint ignores the biology component of the problem field, is founded on data mining algorithms only, and is related in its approach to the exclusion rule generator.

The cross-validations to the Swiss-Prot section of UniProt were sampled as follows. Whenever annotation rules from the individual sources were applied on a Swiss-Prot protein and a predicted annotation was not present in the entry itself, it was considered to be erroneous. Whenever this prediction was contradicted by the Xanthippe system,

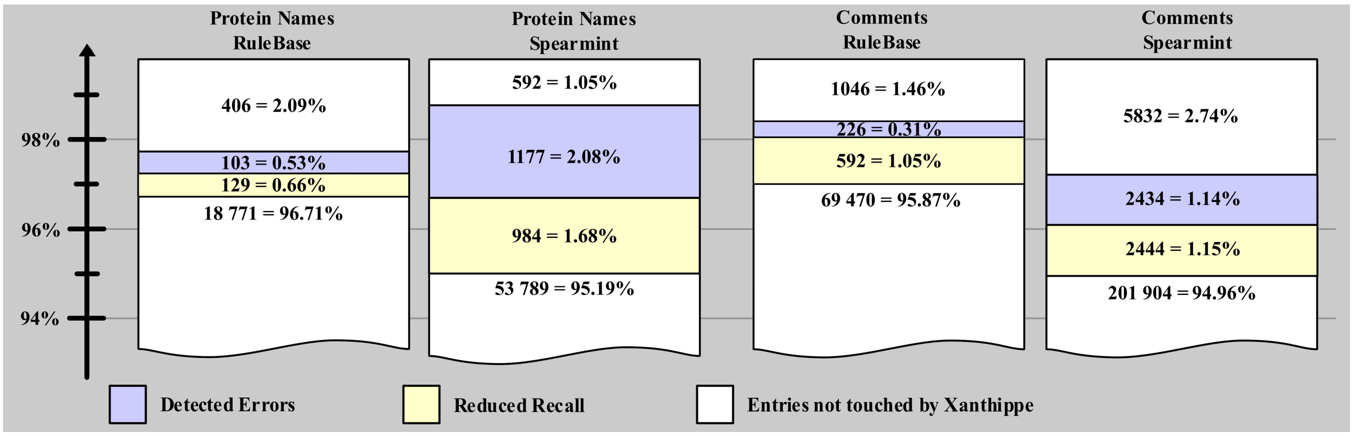

Fig. 5.3: Performance of Xanthippe exclusion trees on comment and protein name predictions from RuleBase and Spearmint.

it was counted as detected (true positive); if it was not contradicted, it was counted as missed (false negative). There are cases where correct annotation was contradicted (false positive), but the largest part consists of correct annotations that were not contradicted (true negative).

Figure 5.2 shows the cross-validation of keyword predictions to Swiss-Prot. For Spearmint, approximately two thirds of annotation errors are detected (blue part) for the price of around two per cent of wrongly contradicted annotations (yellow part). If this validation is true for the target set, it suggests that if Spearmint predictions were applied to the TrEMBL section of UniProt physically rather than by using the current prescriptive annotation system, the quality of keyword predictions could be increased from around 98.5% to ca. 99.5%. For RuleBase still about 40% of the annotation errors were found, improving the overall precision from 99.0% to 99.4%. However, the extended HTML view of a UniProt entry contains notes indicating which annotation is contradicted. Currently neither predicted nor curated annotation is removed from a UniProt entry physically.

Figure 5.3 shows the Xanthippe performance on protein names and comments. For Spearmint, the protein name precision of the predictions could be increased from 96.9% to nearly 99%, all other items showed a poorer performance. After all, between

20% and 30% of the annotation errors could still be filtered on this data, but obviously these are the best targets for further improvements.

Chapter 6

Discussion

The presented system for filtering erroneous annotation from protein entries in UniProt, particularly the automatically produced data, proved useful for keyword annotation. The filtering uses two distinct mechanisms, of which the simple mapping approach was approved to be taken into the automated annotation production pipeline. The exclusion tree approach turned out to be promising, but the performance of comment and protein name contradictions needs further improvement.

Improvements to the exclusion tree approach

Taking sub-string / super-string situations into account is expected to enhance the system considerably. In the InterPro family IPR000500 (*"Connexins"*) for instance, all protein names of the Swiss-Prot members are annotated as *"Gap junction [extension] protein"*, where the extension can be *"alpha-1"*, *"beta-2"* , etc. The RuleBase system however predicts *"Gap junction protein"* without the extension as protein name. Obviously, there can be no Xanthippe rule that ever contradicts this particular annotation, because there is no single instance in the training set where the extension is missing. Should there be a case where RuleBase annotates *"Gap junction protein"* incorrectly, there is nothing in the current Xanthippe system that could prevent this from happening. Furthermore, the statistics given in the above diagrams are impaired by this effect. In the cross-validation, *"Gap junction protein"* predictions on actual *"Gap junction* *[extension] protein"* annotations are still considered as false positives. Mostly, sub-string predictions are generalisations of the actual annotation and should be calculated as true positives. The Xanthippe system needs to be extended to cover sub-strings of protein names and comments, and hence those cases need to be included in the training sets.

Predictors vs. Contradictors

The RuleBase system has been used since 1999 to produce automated annotation. The performance in terms of precision is highly appreciated and the system is supported by a team of experts. Spearmint is intended to supplement RuleBase and to increase its coverage without compromising its level of precision.

As an automatic data mining system Spearmint has the advantage of producing much more annotation than RuleBase. The latter however is small enough to be reviewed constantly by scientists, and hence the confidence in the data it produces is high. The output quality of Spearmint can be adjusted by not applying all the exported rules, but only those with high statistical support. Taking RuleBase as a benchmarking system, Spearmint is set to produce the same quality level as RuleBase. The Spearmint application, followed by a Xanthippe post-processor, can be adjusted to produce even more predictions at a lower level of precision. If a large portion of the errors is filtered out, the same overall annotation quality can be produced as in a more restrictive Spearmint export without the Xanthippe post-processing step.

Running Spearmint at 98.5% quality reproduces 33% of keyword annotation in Swiss-Prot [Kretschmann *et al.*, 2001], while a 95% quality level yields 58% recall. Provided that Xanthippe still detects two thirds of the errors produced by a Spearmint exported at 95%, the actual precision is expected to be at around the desired 98.5%. Additionally, the proportion of detected erroneous annotation is expected rather to increase for a rule set produced by a comparatively low precision system. This means that with using Xanthippe the recall of keywords can be nearly doubled without compromising the quality of the prediction.

Feedback loops

If RuleBase or Spearmint predicts an annotation item on a Swiss-Prot entry, which is not contradicted by Xanthippe but is missing in the entry, a further investigation should be undertaken. In some cases the item might be missing in the entry and can be added, which would improve data consistency in Swiss-Prot.

Since garbage in – garbage out effects are responsible for many annotation errors, the signatures leading to the most obvious ones will be reported to InterPro. If required, such hits can be set to false positive status in this database.

Chapter 7

Conclusion

The aim of this master thesis was to develop a software system that automatically detects erroneous protein annotation. A comprehensive data-mining environment was developed, with which it is possible to perform this task. A CD-ROM with the program code of the Java data-mining application is attached to this master thesis. The results are very promising. A publication is currently in print [Wieser *et al.*,2004] and the work will be presented in August at the ISMB 2004.

Appendix A

The Involved Databases

A.1 UniProt

The UniProt Knowledgebase (UniProt) provides a central database of protein sequences. It holds the Swiss-Prot database, the TrEMBL database and PIR (Protein Information Resource). The Swiss-Prot and TrEMBL section of UniProt were essential parts of this master thesis and they will be introduced in more detail below.

A.2 Swiss-Prot

The Swiss-Prot section of UniProt served as a reference database during this master thesis and delivered the training-proteins for the data-mining. It is a well-annotated protein sequence database established in 1986/87 by the group of Amos Bairoch first at the Department of Medical Biochemistry of the University of Geneva and now at the Swiss Institute of Bioinformatics (SIB) and the European Bioinformatics Institute (EBI). It currently contains over 150 000 protein entries (as per Swiss-Prot Release 43.5 of 07-Jun-2004; http://us.expasy.org/sprot/). The annotation is added manually by a large number of database curators with a strong biological background. Each annotation is double-checked several times in a chain of annotation steps until it is finally accepted by Amos Bairoch and checked into the database. In this way a very high quality of the data could be ensured throughout the years. Swiss-Prot is heavily used as a source for tools and other databases to search for functional groups, protein families or proteins whose sequences show similarities to a given sequence.

A.3 TrEMBL

The TrEMBL section of UniProt was the target database on which the rules that were created after mining the Swiss-Prot database were applied. Before an entry is checked into the Swiss-Prot database, it is usually stored in the TrEMBL database. The TrEMBL database currently contains over 1 000 000 protein entries. The quality and quantity of information assigned to a protein entry in TrEMBL varies. Some entries contain high quality data, while the largest part of TrEMBL contains poorly annotated protein entries, that only provide little more than the mere sequence data. A part of the annotation was imported form the EMBL nucleotide sequence database while a considerably amount of information comes from the automated annotation. The correctness of annotation in the TrEMBL database is not always guaranteed, as the database is rarely checked by human experts. It was the aim of this master thesis to improve the data quantity and in particular the data quality of the entries in the TrEMBL database with the aid of data-mining techniques.

A.4 Structure of Swiss-Prot and TrEMBL

The information about each protein in Swiss-Prot and TrEMBL is collected in a structure called ”Entry”. Two classes of data can be distinguished in an entry: the core data and the annotation. The core data consists of incontrovertible facts like the sequence data, the citation information or the taxonomic data. This information is present for each protein entry in both the Swiss-Prot and the TrEMBL database. This situation is in contrast to the annotation, of which there is a significant amount attached to all Swiss-Prot entries, but which is at least partly unavailable in many TrEMBL protein entries. The annotation is a collection of conclusions made from a scientific point of view and is therefore contingent on the current level of research. It describes the functions of the protein, the similarities to other proteins, or diseases associated with deficiencies in the protein. Each sequence entry is composed of lines. Different types of lines are used to record the various data that make up the entry. Some line types are found in all entries, others are optional, some occur many times in a single entry others at most once. A complete description of all line types is maintained at http://us.expasy.org/sprot/userman.html. The following sections briefly describe those lines that were used in the data-mining application (descriptions are partly extracted from the expasy web page). Firstly, those lines types are introduced that are present for each protein entry in UniProt, i.e. Swiss-Prot and TrEMBL (”Core Data”). The information which they hold were finally used as conditions of the exclusion rules. Secondly, the annotation data items which Xanthippe currently aims to contradict are mentioned (”Annotation”).

ID (Core Data)

The first line of each entry is the ID line which contains the entry name, the data class (STANDARD = Swiss-Prot, PRELIMINARY = TrEMBL), the molecule type (PRT = protein) and the length of the sequence which is given in the number of amino acids:

```
ID CYC_BOVIN STANDARD; PRT; 104 AA.
```

OS (Core Data)

The OS (**O**rganism **S**pecies) line specifies the organism(s) which was (were) the source of the stored sequence.

```
OS Mus musculus (Mouse),
OS Rattus norvegicus (Rat), and
OS Bos taurus (Bovine).
```

OC (Core Data)

The OC (**O**rganism **C**lassification) lines contain the taxonomic classification of the source organism. The classification is listed top-down as nodes in a taxonomic tree in which the most general grouping is given first.

```
OC Eukaryota; Metazoa; Chordata; Craniata; Vertebrata;
OC Euteleostomi;Mammalia; Eutheria; Primates; Catarrhini.
```

DR (Core Data)

The DR (**D**atabase cross-**R**eference) lines are used as pointers to information related to entries and found in data collections other than Swiss-Prot. It holds references to InterPro [Mulder *et al*., 2003], Prosite [Hulo *et al*., 2004], SMART [Letunic *et al*., 2004], TIGRFAMs [Haft *et al*.,2003], PIR Super family [Wu *et al*., 2003] ,
ProDom [Servant *et al*., 2002], Pfam [Bateman *et al*., 2004] and other families to which the protein sequence is assigned. It turned out that the presence or the absence of references to these databases in a protein entry frequently leads to the presence or to the absence of certain annotation data items in this entry. Hence, the DR line is used by the contradiction system Xanthippe to create its exclusion rules. The InterPro database played a specific role in this master thesis and is described in more detail later in this Appendix.

```
DR Pfam; PF00017; SH2; 1.
```

DE (Annotation)

The DE (**DE**scription) lines contain general descriptive information about the sequence stored. This information is generally sufficient to identify the protein precisely. The description line contains three different types of information that can be predicted from the prediction tools Spearmint and RuleBase or rather contradicted form the contradiction system Xanthippe. These are the protein name, synonyms and EC-numbers. It starts with the protein name. Synonyms and EC-numbers are indicated in brackets.

```
DE Annexin V (Lipocortin V) (Endonexin II) (Calphobindin I)
DE (Placental anticoagulant protein I) (PAP-I) (PP4)
DE (Vascular anticoagulant-alpha) (VAC-alpha) (Anchorin CII).
```

KW (Annotation)

The KW (**K**ey **W**ord) lines provide information that can be used to generate indices of the sequence entries based on functional, structural, or other categories. The keywords chosen for each entry serve as a subject reference for the sequence. There is a limited number of keywords (about 800) out of which the database curators choose the most fitting. A list of all currently used Swiss-Prot keywords and a definition of their usage can be found here: http://us.expasy.org/cgi-bin/keywlist.pl.

An example of KW lines in an entry is:

```
KW Apoptosis; Endocytosis; Cell adhesion;
KW Coated pits; Neurone; Heparin-binding;
KW Zinc; Signal; Transmembrane; Glycoprotein;
KW Proteoglycan; Alternative splicing;
KW Alzheimer’s disease; Amyloid; 3D-structure.
```

TrEMBL makes use of the same controlled list of keywords that is used in Swiss-Prot but, as most keywords in an entry are added during the annotation process, TrEMBL entries generally contain fewer keywords than Swiss-Prot entries.

CC (Annotation)

The CC (**C**omments)lines are free text comments on the entry, and are used to convey arbitrary useful information.

```
CC -!- ALLERGEN: Causes an allergic reaction in human.Binds
IgE.
CC It is a partially heat-labile allergen that may cause both
CC respiratory and food-allergy symptoms in patients with the
CC bird-egg syndrome.
```

A.5 InterPro

InterPro [Mulder *et al.*, 2003] is an integrated documentation resource for protein families, domains and sites. It combines several member databases that all hold protein signatures. The member databases use different methods to derive the protein signatures. By uniting those member databases, InterPro capitalizes on their individual strengths, producing a powerful integrated diagnostic tool. The member databases are Prosite, Prints, ProDom, Pfam, SMART and TIGRFAMs. Each protein belonging either to Swiss-Prot or TrEMBL is assigned to one ore more InterPro groups. Currently there exist about 10 000 different InterPro groups. In this master thesis the Swiss-Prot proteins of one InterPro framed a set of training-proteins for the decision tree algorithm.

Appendix B

Detailed Information on Some Important Fixed Classes

The data-mining application is implemented in Java and composed of a fixed part and an adaptable part. The fixed part consists of classes which are not supposed to be changed, configured or adapted from the user as they are fundamental parts of the data-mining application. These fixed classes were assigned to three different packages, which are basically introduced in the following. For further information please browse the Java documentation which can be find on the attached CD-ROM.

B.1 Package datamining

Holds the core classes to establish the Swiss-Prot data-mining framework. It holds the main class and several classes which model different data structures of the data-mining application.

B.1.1 Class Miner

The main class to start the data-mining process on Swiss-Prot data. A mining process is constructed according to the contents of a given configuration file. This process is started, which itself is able to trigger new data-mining sub-processes.

B.1.2 Class MiningProcess

The central process to mine for protein data. Includes all steps from data loading to rule selection.

B.1.3 Class MiningStep

A data-mining process consists of a sequel of data-mining steps. This class handles the common functionality of the data-mining steps, such as logging, reporting and sub-processing. Each MiningStep can potentially create a number of sub-processes (MiningProcesses) with an exchange of some data-mining steps. This construct is designed to mine for items using an extended or modified approach, if the current approach fails.

B.1.4 Class EntrySet

An EntrySet holds a set of PythiaEntries (java objects of protein entries). It is extended by additional information, e.g. information about the source of the protein entries. The EntrySet basically is the protein training-set for the data-mining application.

B.1.5 Class EntrySetPair

An EntrySetPair consists of a training set and a target set. The training set can be used for the mining process itself and the target set for the application of the rules.

B.1.6 Class DataItem

Defines the general structure of data items that describe a protein. DataItems extracted from the training-set are used in the mining process to predict the same DataItems for other proteins. Non-abstract subclasses need to implement the methods of this class.

B.1.7 Class CoreDataItem

The core data items are extracted from the proteins in the training set and modelled in the class CoreDataItems. This data can be used to predict AnnotationDataItem. Discrete values are not supported properly, currently it is only possible to mark whether a core data item is present or absent in a certain protein entry.

B.1.8 Class AnnotationDataItem

An annotation item represents an entity in a Swiss-Prot database entry that can actually be predicted using CoreDataItems. AnnotationDataItems come as simple type-name pairs.

B.1.9 Class MiningTargets

This class is used to define, which AnnotationDataItems are included and excluded from the mining process. It consists of an inclusion- and exclusion-list. All the elements in the inclusion list are mined for, except the ones defined in the exclusion list. There is no consequence, if AnnoationDataItems are contained in the exclusion list but not present in the inclusion list. In this case the AnnotationDataItem is not included in the mining process. Single annotation data items can easily be excluded, if there is an exclusion and a inclusion list. For example, one can specify to include all keywords (inclusion list) except the keyword ”Hypothetical” (exclusion list). The application will mine for all (about 800) keywords except the ”Hypothetical”. The same could be reached in manually adding all keywords which the application is supposed to mine for to the inclusion list , which costs much more effort than simply adding one keyword to an exclusion list.

B.1.10 Interface Report

This interface should be implemented by all classes reporting on the mining process.

B.2 Package datamining.util

Amongst classes and data structures that concern the whole data-mining application, this packages also contains some classes that are not directly part of the data-mining, but are rather used for preparing the output from the data-mining so that the rules finally can be applied. There are also help tools in the package that are useful for preparing the training-set for the actual mining process. The most important classes are shortly introduced in the following.

B.2.1 InterPro2EntrySet

This class is independent from the mining process and it has its own main method. It is used to get Swiss-Prot entries from the database and to store them as serialized objects in local directories. All entries in the given InterPro groups are stored as EntrySets B.1 in the given directories from which they can be fetched during the mining process.

B.2.2 Class Arff

This class forms a data structure for the ARFF (Attribute-Relation File Format) and is used by other packages during the mining process. The Arff class wraps all WEKA classes (open source data-mining software) in an application specific framework. It holds instances, attributes, annotation data items and combines them.

B.2.3 SpearmintExporter and XanthippeExporter

These two classes scan a flat file which contains a list of prediction and contradiction rules. The SpearmintExporter picks out the prediction rules while the XanthippeExporter collects all contradiction rules. They remove all rules that are below a certain confidence value and also rules that do not have a required number of conditions. The result is written into a new flat file. Both classes have an own main routine and are usually used after the data-mining process.

B.2.4 RedundancyRemover

The RedundancyRemover can be used as a post-processing step after the mining process. It is independent from the other classes of the mining application and has its own main routine. The RedundancyRemover scans a given flat file holding a list of rules, removes all duplicated rules and also subrules of more general ones. For instance, a rule that says that no Proteobacteria should have the keyword *Chloroplast* is removed if there is an additional rule that says that no Bacteria should have the keyword *Chloroplast*. If an entry applies to the first rule, it will also apply to the second rule, but not necessarily the other way round. Thus, it is enough to only apply the second rule.

B.2.5 PMMLGenerator

Generates a PMML document (Predictive Model Markup Language) containing contradiction or prediction rules. It translates a given list of InterProTrees B.3 into PMML. Each InterProTree in the given list, acts as a new rule. The accession-number of each rule correlates with the root node of an InterProTree, which is an InterPro-group.

B.2.6 DecisionTreeBuilder

Postprocessing step after the actual mining process.

B.3 Package datamining.tree

Holds all classes concerning the decision tree. It holds a class that models the decision tree itself and different classes that can be used to handle different parts of the tree, for example the nodes or branches.

B.3.1 DecisionTree

Represents a decision tree. A decision tree consists of nodes,which represent the conditions of a prediction or contradiction rule. A decision tree is created for one annotation data item.

B.3.2 InterProTree

An InterProTree is an extension of the DecisionTree. The root node of an InterProTree is always an InterPro group. Several annotation data items can be attached to the same InterProTree. An InterProTree can hold prediction or contradiction rules, but not both. An InterProTree has a type which specifies the type of rules which it holds. A confidence value is assigned to each InterProTree.

Information about additional classes assigned to each of the three packages can be browsed in the Java-doc.

Appendix C

Detailed Information on the Exchangeable Components

C.1 Package: datamining.loading

This package contains an abstract class defining the loading step of the data-mining application and its implementing classes.

ABSTRACT CLASS:

TrainingSetLoader

The abstract class ”TrainingSetLoader” needs to be extended by all classes that load the training-sets used for the data-mining. Subclasses will allow to experiment with training-set variations.

public abstract boolean hasNextEntrySet()	This method checks if there is an EntrySet (see B.1) with which a new program cycle can be started.

public abstract EntrySet getNextEntrySet()	The method getNextEntrySet() delivers an EntrySet (see B.1) at the beginning of each data-mining cycle.

IMPLEMENTING CLASSES:

InterProLoader

The InterProLoader requests a warehouse to get protein entries belonging to a specified InterPro group. A new EntrySet (see B.1) for each InterPro group is created and the InterProLoader passes the EntrySet objects consecutively when the getNextEntrySet() method is called to a mining process.

LocalLoader

The LocalLoader provides access to EntrySets stored as serialized objects locally in possibly various directories on the hard disk. This class will gather all *.obj files in the given directories, try to read them in as EntrySet objects and provide them sequentially when the getNextEntrySet() method is called.

C.2 Package: datamining.preprocessing

This package contains an abstract class defining the pre-processing step of the data-mining application and its implementing classes.

ABSTRACT CLASS:

EntryPreProcessor

Abstract class defining the behaviour of all classes that perform the pre-processing step.

public abstract EntrySet preprocess(EntrySet entrySet)	The abstract method preprocess() is intended to prepare the EntrySet so that it is appropriate for the mining process. The original EntrySet is passed over and a modified EntrySet is returned.

IMPLEMENTING CLASSES

DefaultPreprocessor

Before data-mining begins the EntrySets are preprocessed by deleting or modifying a couple of entries. This class, which is a concrete subclass of the EntryPreProcessor, allows to

- delete all fragments from an EntrySet.
- delete all hypothetical proteins from an EntrySet.
- remove all old RuleBase annotation (evidence tagged) from all entries in an EntrySet.
- remove all old Pythia predictions form all entries in an EntrySet.
- include the keyword hierarchy on all keyword data items (predictions, in original entry) in an EntrySet.

C.3 Package: datamining.splitting

Contains an abstract class defining the splitting step of the data-mining application and its implementing class.

ABSTRACT CLASS:

EntrySetSplitter

The abstract class TrainigSetSplitter has to be extended from classes that aim to split an EntrySet into two smaller EntrySets, i.e. into a training-set and a target-set, for crossvalidation reasons.

public abstract EntrySetPairs splitTrainingSet(EntrySet entrySet)	An EntrySet is passed into the splitting routine. The EntrySet is split into two smaller EntrySets which are saved in an EntrySetPair. The data structure EntrySetPairs hold one ore more EntrySetPair and is returned after splitting the EntrySet in an arbitrary number of EntrySetPair.

IMPLEMENTING CLASSES

DefaultEntrySetSplitter

The DefaultEntrySetSplitter puts the whole EntrySet in the training set part of an EntrySetPair, whereas the target part remains empty. The EntrySetPairs hence is returned with only one element. The whole EntrySet is used for the mining process and no cross-validation is possible with the DefaultEntrySetSplitter.

C.4 Package datamining.arff

ABSTRACT CLASSES:

Contains two abstract classes defining the ARFF-file generation step of the data-mining application and their implementing classes.

ArffPreparator

The ArffPreparator returns an incomplete ARFF-file composed of protein data extracted from all entries of the EntrySet, i.e. it returns the core data and the instances, whereas the core data of the proteins are considered to be attributes like signature hits and the instances are the proteins itself. The ARFF-file is incomplete, because the information which annotation data item the mining process is supposed to mine for, is still missing.

public abstract Arff getCoreDataArff(EntrySet entrySet)	This method sets the EntrySet for which a core data Arff object will be generated. The generated Arff is returned to the calling method.

ArffGenerator

All implementing classes of the ArffGenerator are supposed to return a complete ARFF-File which is ready for the mining process. A prepared core data Arff object is extended with the annotation data item which is supposed to be predicted.

public abstract Arff getCompleteArff(Arff arff, AnnotationDataItem annotationItem)	A prepared core data arff and an AnnotationDataItem are passed into the method. The AnnotationDataItem is attached to the core data Arff object. A complete Arff object is returned, which is used for the following mining steps.

IMPLEMENTING CLASSES

DefaultArffPreparator

The DefaultArffPreparator is a subclass of the ArffPreparator. It extracts all core data from the protein entries that are specified in the configuration files and saves it in an Arff object as well as the protein objects itself.

DefaultArffGenerator

The DefaultArffGenerator completes the prepared Arff-File in adding the annotation data item which the algorithm is supposed to mine for. A new complete ARFF-File is generated for each annotation item.

C.5 Package datamining.mining

Contains one abstract class defining the mining step of the data-mining application and its implementing class.

ABSTRACT CLASS:

DataMiner

The abstract class DataMiner that defines the behaviour of classes used for the data-mining step needs to be extended by all mining algorithms. Different subclasses allow different mining approaches.

public abstract DecisionTree mine(Arff arff)	This method generates a DecisionTree for a given Arff object.

IMPLEMENTING CLASSES

DefaultDataMiner

An ARFF-file which describes the properties of the training-set proteins is passed into the mining routine. A decision tree algorithm scans the Arff object and returns a yes/no decision tree. Therefore a provided implementation of the C4.5 decision tree algorithm of the weka data-mining package is used. The returned decision tree has the form that the Weka package suggests.

C.6 Package datamining.postprocessing

Contains one abstract class defining the post-processing step of the data-mining application and its implementing class.

ABSTRACT CLASS:

DecisionTreePostProcessor

The tree can be processed and redesigned from classes that extend the DecisionTreePostProcessor.

public abstract DecisionTreeSet postprocess(DecisionTree tree, EntrySet entrySet);	The ABSTRACT CLASS: 'DecisionTreePostProcessor' defines the method postprocess, that requires as input the decision tree output of the previous mining step, e.g. the output of the Weka package, and the EntrySet from which the decision tree was created. It is required that a DecisionTreeSet is returned. The DecisionTreeSet holds a collection of decision trees. The implementation of the postprocess method is up to the subclasses of the DecisionTreePostProcessor.

IMPLEMENTING CLASSES

DefaultPostProcessor

The DefaultPostProcessor extends the decision tree that is passed into the postprocessing method. Firstly, it adds initial conditions to it. These are conditions that must be fulfilled by a protein so that it is assigned to an EntrySet. All proteins that belong to IPR000222 may be assigned to an EntrySet. Hence the preconditions for these proteins is to belong to IPR000222. This information is lost in the ARFF-file as it only holds information that belongs to some of the proteins but not if the condition is fulfilled by all proteins. Hence the precondition is missing in the tree. Nevertheless, it belongs to the decision tree and is therefore attached to it as a root node in the postprocess method of the DefaultPostProcessor. The DefaultPostProcessor also adds the protein entries themselves to the branches. Therefore it iterates the EntrySet, steps through the tree for each entry and attaches the entry to the branches whenever it fulfils the conditions. Thus, one can reproduce which proteins of the training-set led to a rule. Finally the DecisionTree is added to a new instance of a DecisionTreeSet, that is returned with only this one element.

SubtreeGenerator

This class post-processes the tree that was generated after mining the ARFF-file. Its postprocess method attaches the initial conditions and the entries to the decision tree. In addition it divides the given tree into several subtrees. This has the advantage that the smaller subtrees are more concise and it makes rule selection easier in the following mining step. All subtrees are added to a DecisionTreeSet. Before the DecisionTreeSet is returned to the calling class of the postprocess method, the SubtreeGenerator attaches security patterns to the sub-trees. It scans all examples in the training-set that fulfil all conditions of a path in a given tree and extracts those patterns which do not happen in the negative examples. The patterns which are specific for the positive examples are attached to the tree. A parameter **postprocessorParameters** can be specified in the configuration file. This specifies the ratio of security patterns, i.e. if the parameter is set to 0.8, the pattern must occur at least in 80% of the positive examples, while it must not occur more frequently than in 20% of the negative examples.

C.7 Package datamining.selecting

Contains one abstract class defining the selecting step of the data-mining application and its implementing class.

ABSTRACT CLASS:

DecisionTreeSelector

public abstract DecisionTreeSet select(DecisionTreeSet trees)	This method selects a set of trees with a certain reliability from a given set of trees.

IMPLEMENTING CLASSES

DefaultSelector

The DefaultSelector returns the DecisionTreeSet as it is, i.e. it keeps all generated rules and no rules are cancelled.

C.8 Package datamining.crossvalidation

Contains one abstract class defining the cross-validation step of the data-mining application and its implementing class.

ABSTRACT CLASS:

DefaultCrossValidator

public abstract void cross-validate(DecisionTreeSet decisionTreeSet, EntrySet targetset, MiningTargets miningTargets)	Applies a given set of trees on target proteins and possibly analyzes the result.

IMPLEMENTING CLASSES

DefaultCrossValidator

The DefaultCrossValidator is used if no crossvalidation of the rules is required, as it keeps the variables that were passed over unchanged.

StandardCrossValidator

The StandardCrossValidator iterates the given DecisionTreeSet that holds the rules. It takes one tree after the other and iterates the EntrySet to determine which of the entries fulfill the conditions of the rules. Those rules that fulfill the condition are attached to the decision tree.

C.9 Package datamining.exporter

Contains one abstract class defining the exporting step of the data-mining application and its implementing class. The classes in the ”Exporter” package intend to save the final rules that were generated in the mining process.

ABSTRACT CLASS:(es)

Exporter

The implementing classes of this class intend to save the final rules that were generated in the mining process.

public abstract void saveRules(DecisionTreeSet treeSet)	Saves rules of a given set of trees, possibly after a selection process.

IMPLEMENTING CLASSES

DefaultExporter

Saves all rules that can be extracted from the given tree set into one file.

ClientExporter

Saves all rules that can be extracted from the given tree set into one file. The file name is assigned automatically.

Bibliography

[Apweiler *et al.*, 2004] Apweiler,R., Bairoch,A., Wu,C.H., Barker,W.C., Boeckmann,B., Ferro,S., Gasteiger,E., Huang,H., Lopez,R., Magrane,M., Martin,M.J., Natale,D.A., O'Donovan,C., Redaschi,N. and Yeh,L.S. (2004) UniProt: the Universal Protein Knowledgebase. *Nucleic Acids Res.*, **3**2, D115–D119.

[Attwood *et al.*, 2003] 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.*, **3**1(1), 400–402.

[Bateman *et al.*, 2004] Bateman A, Coin L, Durbin R, Finn RD, Hollich V, Griffiths-Jones S, Khanna A, Marshall M, Moxon S, Sonnhammer EL, Studholme DJ, Yeats C, Eddy SR. (2004) The Pfam protein families database.*Nucleic Acids Res.*, **3**2, 138–41.

[Biswas *et al.*, 2001] Biswas,M., O'Rourke,J.F., 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. *Briefings in Bioinformatics*, **3**(3), 285–295.

[Breiman *et al.*, 1984] Breiman,L., Friedman,J.H., Olshen,R.A. and Stone,C.J. (1984) Classification and Regression Trees, Wadsworth: Belmont, CA.

[Fleischmann *et al.*, 1999] Fleischmann,W., Mller,S., Gateau,A., Apweiler,R. (1999) A novel method for automatic functional annotation of proteins. *Bioinformatics*, **1**5(3), 228–33.

[Grossman *et al.*, 1999] R. L. Grossman, S. Bailey, A. Ramu and B. Malhi, P. Hallstrom, I. Pulleyn and X. Qin (1999) The Management and Mining of Multiple Predictive Models Using the Predictive Modeling Markup Language (PMML), **I**nformation and Software Technology.

[Haft *et al.*,2003] Haft DH, Selengut JD, White O. (2003) The TIGRFAMs database of protein families.*Nucleic Acids Res.* **3**1, 371–373.

[Hulo *et al.*, 2004] Hulo N, Sigrist CJ, Le Saux V, Langendijk-Genevaux PS, Bordoli L, Gattiker A, De Castro E, Bucher P, Bairoch A. (2004) Recent improvements to the PROSITE database. *Nucleic Acids Res.*, **3**2, 134–7.

[Kretschmann *et al.*, 2001] 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*, **1**7, 920–926.

[Kulikova *et al.*, 2004] Kulikova,T., Aldebert,P., Althorpe,N., Baker,W., Bates,K., Browne,P., Van Den Broek,A., Cochrane,G., Duggan,K., Eberhardt,R., Faruque,N., Garcia-Pastor,M., Harte,N., Kanz,C., Leinonen,R., Lin,Q., Lombard,V., Lopez,R., Mancuso,R., McHale,M., Nardon,F., Silventoinen,V., Stoehr,P., Stoesser,G., Tuli,M.A., Tzouvara,K., Vaughan,R., Wu,D., Zhu,W. and Apweiler,R. (2004) The EMBL Nucleotide Sequence Database. *Nucleic Acids Res.*, **3**2, D27–D30.

[Letunic *et al.*, 2004] Letunic I, Copley RR, Schmidt S, Ciccarelli FD, Doerks T, Schultz J, Ponting CP, Bork P. (2004) SMART 40: towards genomic data integration. *Nucleic Acids Res.* **3**2, 142–4.

[Mulder *et al.*, 2003] Mulder,N.J., Apweiler,R., Attwood,T.K., Bairoch,A., Barrell,D., Bateman,A., Binns,D., Biswas,M., Bradley,P., Bork,P., Bucher,P., Copley,R.R., Courcelle,E., Das,U., Durbin,R., Falquet,L., Fleischmann,W., Griffiths-

Jones,S., Haft,D., Harte,N., Hulo,N., Kahn,D., Kanapin,A., Krestyaninova,M., Lopez,R., Letunic,I., Lonsdale,D., Silventoinen,V., Orchard,S.E., Pagni,M., Peyruc,D., Ponting,C.P., Selengut,J.D, Servant,F., Sigrist,C.J.A., Vaughan,R. and Zdobnov,E.M. (2003) The InterPro Database, 2003 brings increased coverage and new features. *Nucleic Acids Res.*, **3**1, 315–318.

[Prlic *et al.*, 2004] Prlic,A., Domingues,F.S., Lackner,P. and Sippl,M.J. (2004) WILMA - Automated annotation of protein sequences. *Bioinformatics*, **2**0(1), 127–128.

[Quinlan, 1993] Quinlan,J.R. (1993), C4.5:Programs for Machine Learning. Morgan Kaufmann, San Francisco, CA.

[Schultz *et al.*, 2000] Schultz,J., Copley,R.R., Doerks,T., Ponting,C.P. and Bork,P. (2000) SMART: A Web-based tool for the study of genetically mobile domains. *Nucleic Acids Res.*, **2**8, 231–234.

[Servant *et al.*, 2002] Servant F, Bru C, Carrere S, Courcelle E, Gouzy J, Peyruc D, Kahn D. (2002) ProDom: Automated clustering of homologous domains. *Briefings in Bioinformatics*,**3**, 246–251.

[Wieser *et al.*,2004] Wieser D, Kretschmann E, and Apweiler R. (2004) Filtering Erroneous Protein Annotation *Bioinformatics*. In print.

[Witten *et al.*,2000] Ian H. Witten and Eibe Frank (2000) Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.

[Wu *et al.*, 2003] Wu CH, Yeh LL, Huang H, Arminski L, Castro-Alvear J, Chen Y, Hu Z, Ledley RS, Kourtesis P, Suzek BE, Vinayaka CR, Zhang J, Barker WC. (2003) The Protein Information Resource. *Nucleic Acids Res.* **3**1, 345–347.

[Zdobnov *et al.*, 2001] Zdobnov,E.M. and Apweiler,R. (2001) InterProScan - an integration platform for the signature-recognition methods in InterPro. *Bioinformatics*, **1**7, 847–848.