Machine learning and data mining for yeast functional genomics

Amanda Clare

Department of Computer Science
University of Wales
Aberystwyth

February 2003

This thesis is submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy of The University of Wales.

Declaration

This thesis has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed ..... (candidate)

Date .....

Statement 1

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed ..... (candidate)

Date .....

Statement 2

I hereby give consent for my thesis, if accepted, to be made available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed ..... (candidate)

Date .....

Abstract

This thesis presents an investigation into machine learning and data mining methods that can be used on data from the *Saccharomyces cerevisiae* genome. The aim is to predict functional class for ORFs (Open Reading Frames) whose function is currently unknown.

Analysis of the yeast genome provides many challenges to existing computational techniques. Data is now available on a genome-wide scale from sources such as the results of phenotypic growth experiments, microarray experiments, sequence characteristics, secondary structure prediction and sequence similarity searches. This work builds on existing approaches to analysis of ORF function in the *M. tuberculosis* and *E. coli* genomes and extends the computational methods to deal with the size and complexity of the data from yeast.

Several novel extensions to existing machine learning algorithms are presented. These include algorithms for multi-label learning (where each example belongs to more than one possible class), learning with both hierarchically-structured data and classes, and a distributed first order association mining algorithm for use on a Beowulf cluster. We use bootstrap techniques for sampling when data is sparse, and consider combinations of data and rulesets to increase accuracy of results. We also investigate the standard methods of clustering of microarray data, and quantitatively evaluate their reliability and self consistency.

Accurate rules have been learned and predictions have been made for many of the ORFs of unknown function. The rules are understandable and agree with known biology. All predictions are freely available from http://www.genepredictions.org, all datasets used in this study are freely available from http://www.aber.ac.uk/compsci/Research/bio/dss/ and software for relational data mining is available from http://www.aber.ac.uk/compsci/Research/bio/dss/polyfarm.

Acknowledgements

This thesis would not have been possible without the help and support of many people.

Firstly I would like to thank my supervisor, Prof. Ross D. King, for his excellent supervision, his knowledge, his belief and interest in the work and encouragement and motivation throughout. I would also like to thank Andreas Karwath for being my good friend, my mentor and someone to follow.

I am very grateful to David Enot for his hours of patient and detailed proofreading, and many conversations.

My thanks also go to everyone who has provided support or advice in one way or another, including sysadmins, secretaries, PharmaDM, Julian for Haskell support, Eddie for proofreading, the examiners Jem Rowland and Ashwin Srinivasan for their comments, and others. The MRC provided financial support for the work under grant number G78/6609.

Many friends in Aberystwyth have made the past 3 years extremely enjoyable, especially all the past and present residents of E51, everyone in the Computer Science department, my flatmates and Paul.

And finally I'd like to thank my family for their love and support.

Contents

Introduction

The research in this thesis concentrates on the area of functional genomics - elucidating the biological functions of the parts of a genome. When a genome is sequenced, and we have the predicted locations of the genes within the genome, the next stage is to work out the possible functions of these genes. This thesis investigates how to use data mining and machine learning to make predictions for the function of genes in the yeast *Saccharomyces cerevisiae* and to acquire new scientific knowledge and understanding about biological function. The outcomes of this thesis are twofold:

1. Machine learning and data mining research

   This is a challenging environment for machine learning and data mining, and specific challenges are:

   - Use of more of the full range of data available from biology - many new techniques in biology are providing data on a genome wide scale. This data is noisy and heterogeneous.
   - Use of multiple labels - each gene can have more than one function, which is an unusual machine learning environment.
   - Use of hierarchical class information - the biological functions we wish to predict are organised in a hierarchical manner.
   - Scaling problems due to the volume of data in bioinformatics - ever increasing volumes of complex data demand scalable solutions. The methods applied and developed should scale as far as possible to larger genomes for future use.
   - Reliable and accurate results - results require careful validation, and results from different data sources and algorithms need to be appropriately combined and evaluated.

2. Gene function prediction and scientific discovery

   We produce predictions of gene function that are understandable and easily accessible to biologists. These can provide a insight into the biological rea-

sons for the predictions and hence a basis for scientific discovery and new understanding of biology.

The organisation of this thesis will be as follows:

- Chapter 1 introduces the computational background to this thesis - the data mining and machine learning techniques that will be applied and built upon.
- Chapter 2 introduces the fields of computational biology and functional genomics, and the datasets that we will be working with.
- In Chapter 3 we describe a method previously used for prediction of gene function using machine learning and data mining. This method does not scale to all the data we have available for yeast, and we describe the problems that need to be solved.
- Chapter 4 deals with the analysis of yeast phenotype data. The challenges of multi-label data and too many classes with too little data are faced in order to learn from phenotype growth experiment results. An extension to the popular C4.5 algorithm is proposed and implemented. Bootstrap resampling is used to extract the most reliable results.
- Chapter 5 investigates ways to use yeast expression data. The data is noisy and real-valued, and consists of short time-series.
- In Chapter 6 a distributed first order association mining algorithm (PolyFARM) is designed and implemented. This is applied to the relational data sets of yeast homology and secondary structure data.
- Hierarchical data is considered in Chapter 7. The yeast data provides us with both hierarchically structured attributes and a hierarchical class structure. C4.5 is extended to make use of the hierarchical class structure and PolyFARM is extended to make use of the hierarchical attributes. Both algorithms are tested on the yeast data.
- Chapter 8 describes the data that has been collected and presents the overall results from use of all the yeast data sets. Strategies for combination of results are compared.
- Finally Chapter 9 draws conclusions, presents ideas for future work and summarises the original contributions to knowledge that this thesis has made.

Chapter 1

An overview of data mining and machine learning

1.1 The data explosion

Over the past few years there has been an explosion of data in the field of biology. New techniques now make sequencing of whole genomes possible. The complete genetic code of organisms from bacteria and viruses to humans can now be mapped, sequenced and analysed. Along with the rise in genomic data is a huge increase in other biological data from the proteome, metabolome, transcriptome, combinatorial chemistry, etc. The scale of the excitement about the potential of this data is matched by the scale of the resources used to produce it - thousands of machines producing terabytes of data. Analysis of this data is now the key problem and computers are an essential part of this process. Data mining and machine learning techniques are needed which can scale to the size of the problems and can be tailored to the application of biology.

1.2 Data mining and machine learning

Data mining, or knowledge discovery in databases, is the process of extracting knowledge from large databases. Agrawal *et al.* (1993) describe three types of knowledge discovery: classification, associations and sequences.

Classification attempts to divide the data into classes. A characterisation of the classes can then be used to make predictions for new unclassified data. Classes can be a simple binary partition (such as "is-an-enzyme" or "not-an-enzyme"), or can be complex and many-valued such as the classes in our gene functional hierarchies.

Associations are patterns in the data, frequently occurring sets of items that belong together. For example "pasta, minced beef and spaghetti sauce are frequently found together in shopping basket data". Associations can be used to define association rules, which give probabilities of inferring certain data given other data, such as "if someone buys pasta and minced beef then there is a 75% likelihood they also buy spaghetti sauce".

Sequences are knowledge about data where time or some other ordering is involved, for example, to extract patterns from stock market data or gene sequence motifs.

Due to the volume of data in modern databases, data mining by hand is nearly impossible, and machine learning methods are usually used for data mining. Machine learning is a way of automatically improving, of using "training" data to build or alter a model which can later be used to make predictions for new unseen data (Mitchell, 1997).

There are hundreds of machine learning algorithms available now, each with their own characteristics. Some of the dimensions of machine learning algorithms which are useful to consider in this work are:

Supervised vs unsupervised: A supervised algorithm is first trained on a set of labelled data. A set of data whose classes are already known is used to build up profiles of the classes, and this information can then be used to predict the class of new data. Unsupervised learning has no training stage, and is usually used when the classes are not known in advance. Clustering is an example of unsupervised classification. In this case, data which is similar is clustered together to form groups which can be thought of as classes. New data is classified by assignment to the closest matching cluster, and is assumed to have characteristics similar to the other data in the cluster.

Intelligible vs black box: Some machine learning algorithms produce human-readable results whereas others are "black boxes", whose working and intuition cannot be understood. Neural networks and support vector machines are examples of black boxes. However they are often highly accurate in their results, particularly on continuous real-valued numeric data. In this work, we prefer learners which have more readily understandable output, since the results are for biologists, and are for scientific knowledge discovery (Srinivasan, 2001; Michie, 1986; Langley, 1998). It is not enough just to be able to predict, for example, that a gene is involved in protein synthesis, we want to be able to understand why the learner has come to that conclusion.

Missing data/noise: Our data is real-world data, hence it will be noisy and suffer from missing values. It will be important that the machine learner can handle this.

Propositional vs first order: Propositional algorithms assume data can be represented in a simple attribute-value format (for example: `gene_length = 546, lysine_ratio = 4.5, susceptibility_to_sorbitol = yes`). They are generally very fast and efficient with memory usage. First order learning algorithms can handle more complex relational data which cannot naturally be expressed in a propositional form. They can easily express relationships such as gene sequence similarity (for example: `similar(A,B) and classification(B, saccharomyces) and molecular_weight(B, heavy) and length(A, long)` describes a long sequence A which is similar to another heavy *Saccharomyces* sequence B).

Background knowledge: Background knowledge can be encoded in some machine learning algorithms. This is useful to constrain the search space, produce less of the uninteresting solutions, and make better use of all available information. Srinivasan *et al* (1999) demonstrate the benefits of using background knowledge in machine learning in a chemistry domain.

Continuous vs symbolic attributes: Some machine learning algorithms are much better with continuous numeric data, and others prefer symbolic data. In this work we have a mixture of both types of data, and this will sway the choice of algorithm. If a specific algorithm is desired which is not good with continuous data, it may be necessary to convert the continuous data into discretised data.

The next few sections will describe various machine learning algorithms which will be used in this work.

1.3 Decision trees

Decision tree algorithms are supervised algorithms which recursively partition the data based on its attributes, until some stopping condition is reached. This recursive partitioning gives rise to a tree-like structure. The aim is that the final partitions (leaves of the tree) are homogeneous with respect to the classes, and the internal nodes of the tree are decisions about the attributes that were taken to reach the leaves. (As a counterpart to decision trees, clustering trees also exist, where internal nodes can also be thought of as classes). The decisions are usually simple attribute tests, using one attribute at a time to discriminate the data. New data can be classified by following the conditions at the nodes down into a leaf. Figure 1.1 shows an example of a decision tree that chooses an appropriate form of transport, given attributes about the weather conditions and transport availability.

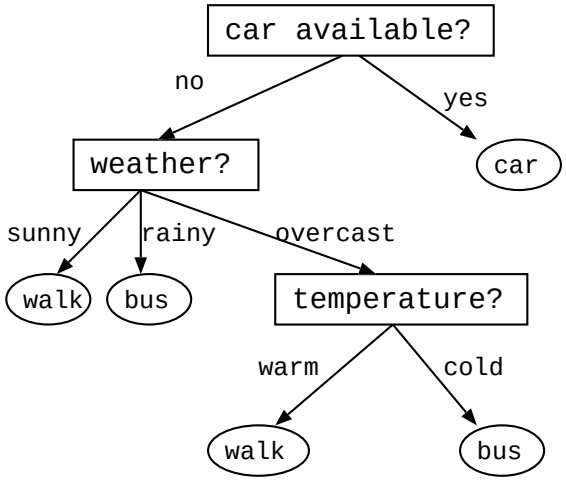


Figure 1.1: A simple decision tree, showing decisions at the nodes, and final classification at the leaves

If the attribute used to make the partition is symbolic, then there will be one branch per possible value of the attribute. If the attribute is continuous, the branch will usually be a two-way choice: comparing the value to see whether it is less than a fixed constant or not. This constant is determined by the range of values in the dataset.

The criterion for choosing the best attribute for the decision at each node varies from algorithm to algorithm. Examples of such measures include information gain of the split, the Gini measure (a measure of the impurity of a node, calculated by $1 - \sum_j p(j|t)$ where $p(j|t)$ is the probability of class j at node t), and the Twoing rule which tries to balance the number of items on each side of the split.

Algorithms also vary on their stopping and pruning criteria: how they decide when to stop growing the tree and how far back to prune it afterwards. We want a tree that can be general enough to apply to new datasets, and most data has noise and classes which cannot easily be learned. The right level of detail must be found so as not to overfit the tree to the data.

Decision tree algorithms are very efficient, which is desirable for large volumes of data. This is due to the partitioning nature of the algorithm, each time working on smaller and smaller pieces of the dataset, and the fact that they usually only work with simple attribute-value data which is easy to manipulate. One of their drawbacks is that the divisive partitioning can cause data with interesting relationships to be separated right from the start, and there is no way of recovering from these mistakes later on.

CART (Classification and Regression Trees) (Breiman *et al.*, 1984) is a decision tree package that includes seven single variable splitting criteria. These are Gini,

symmetric Gini, twoing, ordered twoing and class probability for classification trees, and least squares and least absolute deviation for regression trees, and also one multi-variable splitting criteria, the linear combinations method. Now sold by Salford Systems, this package has been very successful in a variety of applications and won the KDDCup 2000 web mining competition (Kohavi *et al.*, 2000).

Possibly the best known decision tree algorithm, and the one we will work with, is C4.5 (and its commercial successor C5.0) (Quinlan, 1993). It is easy to use "off the shelf", it is reasonably accurate and it has been so successful that it is often reported by the machine learning community as the standard baseline algorithm against which to compare their current algorithms. C4.5/C5.0 uses entropy as the measure on which to partition the data and later prune the tree.

1.4 Clustering

Clustering is an unsupervised machine learning technique. A similarity metric is defined between items of data, and then similar items are grouped together to form clusters. Properties about the clusters can be analysed to determine cluster profiles, which distinguish one cluster from another and say something about the data that share a cluster. New data can be classified by placing it in the cluster it is most similar to, and hence inferring properties from this cluster.

There are many ways of building clusters (Jain *et al.*, 1999; M. Steinbach *et al.*, 2000; Fasulo, 1999; Jain & R. C. Dubes, 1988). **Agglomerative** clustering starts with single items and joins them together to make small clusters. The small clusters are joined again to make larger clusters, which can in turned be joined, and so on. **Divisive** or **partitional** clustering works the opposite way, top-down, like decision trees, partitioning the data into smaller and smaller clusters each time. **Exclusive** clustering allows each data item to belong only to one cluster, whereas **non-exclusive** clustering allows it to be assigned to several clusters, and **probabilistic** clustering allows an item to belong to a cluster with a certain probability. A clustering can be **hierarchical** or **non-hierarchical**, and other aspects of clustering could allow the algorithm to be partially supervised by making use of given class information, or only use a single attribute at a time.

We will use a few of the most standard and well known clustering algorithms: agglomerative hierarchical clustering and k-means clustering. Other methods available include self-organising maps, nearest neighbour, and growing networks.

Agglomerative hierarchical clustering works by the repeated joining of small clusters to make larger clusters, finally producing a hierarchy in which the leaves are the individual data items, and the nodes represent the joining of their children clusters. The hierarchy can be grown until there is a single root node, or until some stopping criterion is reached. Any level in the hierarchy represents a particular level of cluster granularity or detail. When deciding whether to merge two clusters,

there are several strategies which can be employed. Single linkage will consider the strongest single similarity between any two items in the clusters, complete linkage will consider the weakest single similarity, and average linkage will consider the average of the similarities between all items.

K-means clustering, on the other hand, has no concept of a hierarchy. A fixed number of cluster centres (k) is chosen in advance, and data items are assigned to their nearest cluster centre. A recalculation of the cluster centres is done, based on the items belonging to each cluster, and then the data is reassigned to the new clusters. This is repeated until some stopping criterion is achieved (such as no more change).

The validity of the clustering can be measured in several ways: by reference to some known intuition or facts about the data (such as actual class labels which were not used during the clustering), by considering inter-cluster density and intra-cluster compactness, or by using a relative approach of comparing it to other clustering schemes.

1.5 Association rule mining

1.5.1 Introduction

Association rule mining is a common data mining technique which can be used to produce interesting patterns or rules. Association rule mining programs count frequent patterns (or "associations") in large databases, reporting all that exist above a minimum frequency threshold known as the "support". Association rule mining is a well established field and several surveys of common algorithms exist (Hipp *et al.*, 2000; Mueller, 1995). The standard example used to describe this problem is that of analysing supermarket basket data, where a supermarket would want to see which products are frequently bought together. Such an association might be "if a customer buys minced beef and pasta then they are 75% likely to buy spaghetti sauce".

Some definitions follow:

1. An **association** is any subset of items (for example $\{pasta, mince, sauce\}$)
2. The **support** of an association X is the relative frequency of X in the database. If there are 10 examples in the database containing the association $\{pasta, mince\}$, and the database has 1000 examples in it, then the support of $\{pasta, mince\}$ would be 0.01.
3. The **confidence** of a rule $X \rightarrow Y$ is $\frac{support(X \cup Y)}{support(X)}$
4. An association is **frequent** if its support is above the predefined minimum support threshold

5. A rule **holds** if its confidence is above the predefined minimum confidence threshold and its support above the predefined minimum support threshold

The amount of time it takes to count associations in large databases has led to many clever algorithms for counting, and investigation into aspects such as minimising I/O operations to read the database, minimising memory requirements and parallelising the algorithms. Certain properties of itemsets are useful when minimising the search space. Frequency of itemsets is **monotonic**: if an itemset is not frequent, then no specialisations (supersets) of this itemset are frequent (if $\{pasta\}$ is not frequent, then $\{pasta, mince\}$ cannot possibly be frequent). And if an itemset is frequent, then all of its subsets are also frequent.

1.5.2 AIS

AIS (Agrawal & Srikant, 1994) was one of the early association rule mining algorithms. It worked on a levelwise basis, guaranteeing to take at most $d + 1$ passes through the database, where d is the maximum size of a frequent association. First a pass is made through the database, where all singleton itemsets are discovered and counted. All those falling below the minimum support threshold are discarded. The remaining sets are called "frontier sets". Next, another pass through the database is made, this time discovering and counting frequencies of possible 1-extensions which can be made to these frontier sets by adding an item. For example, $\{pasta\}$ could be extended to give $\{pasta, mince\}$, $\{pasta, sauce\}$, $\{pasta, wine\}$. Extensions are only made if they actually appear in the database, and are not constructed until encountered while passing through the database. Duplicate associations created in two different ways are avoided by using an ordering of the items, and only extending a set with items that are greater than the last item added. Again, sets with less than the minimum support after the pass through the databases are discarded. The remaining sets become the new frontier sets, and the next level of extensions are made and counted, and so on, until no more extensions can be made.

1.5.3 APRIORI

Perhaps the best known association rule mining algorithm is APRIORI (Agrawal & Srikant, 1994). APRIORI was also one of the early algorithms, but its method of generating associations to count (APRIORI_GEN) was so efficient that almost every algorithm has used it since. It uses the same levelwise approach of AIS, but, at each level, separates the candidate generation stage from the counting stage. APRIORI_GEN tries to cut down on the number of associations to count, by noting that if an itemset is frequent then all of its subsets are frequent, and as its subsets will necessarily be shorter, they will have been already been counted.

So an extension of an itemset can be constructed as follows. Take two frequent frontier sets that differ in one item (ie if they are of size n, then they share n-1 items in common). For example $\{a, b, c, d\}$ and $\{a, b, c, e\}$. Take the union of the two sets ($\{a, b, c, d, e\}$). This new set is a candidate to be counted if all of its subsets are frequent. We know already that two of its subsets are frequent (namely the two that were used to generate this one). So it just remains to check the others ($\{a, b, d, e\}, \{a, c, d, e\}, \{b, c, d, e\}$). Using APRIORI_GEN means that fewer candidate sets need to be counted, which saves time.

1.5.4 PARTITION

PARTITION (Sarasere *et al.*, 1995) was an algorithm which attempted to improve on the amount of time spent in I/O operations reading the database, and to allow the database to be pruned so as not to keep re-reading infrequent data. It required only 2 passes through the database (as opposed to the previous $d + 1$).

The database is split into equal sized chunks, each of which is small enough to entirely fit into main memory. Each chunk is processed separately, and all associations found within it that are above the adjusted minimum support threshold are reported. The minimum support threshold must be adjusted for the new size of the chunk. If the minimum support was S for the whole database, then each piece of a database split into N equal sized pieces would require a minimum support threshold of S/N. If an association is frequent in the whole database, then it must be locally frequent (above the adjusted minimum support) in at least one chunk of the database. So we know that the only associations which can possibly be globally frequent are those that are locally frequent somewhere. A second pass through the whole database is then needed, to recount all associations that were found to be locally frequent in at least one chunk. These are the final associations to be reported.

As the database chunks are small enough to fit into main memory, they can be pruned in memory as the algorithm progresses. The data is stored in an inverted representation, mapping from items to transactions rather than the other way round. For example, see Table 1.1.

item	basket numbers
pasta	1,3,4,6
sauce	1,2,3
mince	1,3,5,6
bread	1,2

Table 1.1: Level 1 associations in PARTITION using the inverted database representation in main memory

It can quickly be seen from this table that "bread" has a support of 2 and "sauce" has a support of 3. Items with less than the minimum support can easily be removed. New extensions to the associations can be made by standard database merge-join operations on this data structure. Table 1.2 shows the associations of this example at the next level, after merge-join operations.

item	basket numbers
pasta,sauce	1,3
pasta,mince	1,3,6
mince,sauce	1,3
bread,sauce	1,2
bread,pasta	1
bread,mince	1

Table 1.2: Level 2 associations in PARTITION using the inverted database representation in main memory

Mueller (1995) gives a summary of many of the basic algorithms and some further work on data structures and parallelisation. He notes that the inverted data representation used in PARTITION did not lead to better performance, and non-partitioned algorithms performed better than partitioned algorithms.

1.5.5 Parallel association rule mining

As the size of data to be mined has increased, algorithms have been devised for parallel rule mining, both for machines with distributed memory (Park *et al.*, 1995b; Agrawal & Shafer, 1996; Cheung *et al.*, 1996; Han *et al.*, 1997) ("shared-nothing" machines), and, more recently, for machines with shared memory (Parthasrathy *et al.*, 2001). These algorithms have introduced more complex data representations to try to speed up the algorithms, reduce I/O and use less memory. Due to the size and nature of this type of data mining, it is often the case that even just keeping the candidate associations in memory is too much and they need to be swapped out to disk, or recalculated every time on the fly. The number of I/O passes through the database that the algorithm has to make can take a substantial proportion of the running time of the algorithm if the database is large. Parallel rule mining also raises issues about the best ways to partition the work.

This type of rule mining is of specific interest to us because we have a Beowulf cluster of machines in the department that can be used to speed up our processing time. This cluster is a network of 64 shared-nothing machines each with its own processor and memory, with one machine acting as scheduler to farm out portions of work to the others.

Shared-nothing machines

For parallel mining on shared-nothing machines Park *et al.* (1995b) proposed the **PDM** algorithm which is an adaptation of DHP (Park *et al.*, 1995a). DHP uses a hash table to reduce the number of candidates generated at each level. The hash table for level k is built in the previous pass for level $(k-1)$. The hash filtering reduces the size of the candidate set, particularly at lower levels (k=2). When parallelised, the hash tables collected for each part of the database need to be communicated between all the nodes, and this causes a large amount of communication. The authors devised a *clue-and-poll* mechanism to deal with this. Only the larger counts are exchanged first, then others can be requested later.

Agrawal and Shafer (1996) investigate three ways of parallelising the APRIORI algorithm. These are:

- **Count Distribution** - avoids communication by doing redundant computation in parallel. In this method for each level k, each node independently and redundantly computes all the new candidates at that level. The candidate set will be the same for each node. Then each node counts candidates on its portion of the database, then exchanges counts with all other nodes. Each node now independently and redundantly prunes the candidates (all nodes have exactly the same set), and goes on to generate the next level of candidates.

- **Data Distribution** - is a "communication-happy" strategy. Each processor counts only a subset of the candidates, but counts them on the whole database, by using its own portion of the database data and portions of the database broadcast from other nodes.

- **Candidate Distribution** - both data and candidates are partitioned. One of the above two algorithms is used initially. Then after a number of passes (determined heuristically) when the conditions have become more favourable, the candidates are partitioned, and the database also partitioned (with duplication of some parts where necessary) so that each node can continue independently.

They discovered that the Count Distribution algorithm worked best, and the Data Distribution algorithm worst due to its communication overhead. They state that "While it may be disheartening to learn that a carefully designed algorithm like Candidate can be beaten by a relatively simpler algorithm like Count, it does at least illuminate the fact not all problems require an intricate parallelization".

Cheung *et al.* (1996) developed the **DMA** algorithm for distributed databases. This has the motivation that data to be mined my be held in several distributed databases (for example in a nation-wide supermarket chain), and the individual

databases may suffer from data skew in that patterns in one database may be quite different to patterns in another. They allow each node to generate its own set of candidates, prune these, and then collect support for these from other nodes, so dividing this work amongst the nodes.

Han *et al.* (1997) claim that Agrawal and Shafer's Count Distribution algorithm is not scalable with respect to increasing candidate size, and instead, introduce two algorithms that improve on their Data Distribution algorithm. These are **Intelligent Data Distribution** and **Hybrid Distribution**. Intelligent Data Distribution uses a ring-based all-to-all broadcast. When the total number of candidates falls below a threshold, the algorithm uses Count Distribution instead. They use intelligent partitioning of candidate sets. Hybrid Distribution improves on this further by dynamically grouping nodes and partitioning the candidate set.

Shared memory machines

The recent work of Parthasrathy *et al.* (2001) is the first to look at parallel association rule mining on a shared memory machine. Their algorithm is based on APRIORI, and they consider the alternatives of partitioning the candidates amongst the processors, or partitioning the database. They also investigated several optimisations, and a memory allocation scheme to improve locality and reduce false sharing (problems when two different shared variables are located in the same cache block and required by different processors).

1.6 Inductive Logic Programming

ILP refers to the collection of machine learning algorithms which use first order logic as their language, both for data input and result output (and often the intermediate stages too). Data is no longer restricted to being represented as attribute-value pairs as in traditional propositional learning algorithms, but predicates and relations can be defined. C4.5 and other "zeroth order" algorithms will require that each item of data has the same number of attributes, which can lead to some convoluted data representations to ensure this. However, ILP algorithms generally allow arbitrary numbers of predicates per example, and also allow background knowledge for data to be expressed in a more convenient form. ILP algorithms have the disadvantage of searching a wider space of hypotheses, and can often be slow and more memory intensive.

ILP algorithms usually use a language bias that allows the user to restrict the search space and direct the algorithm. This often takes the form of declaring types and modes for the predicates that are to be used, along with other constraints such as how many times they can be used, and in which conjunctions with other predicates. This means that some concepts can no longer be expressed in this

language (or it will be awkward to do so), and so the search is restricted.

1.6.1 ALEPH/PROGOL

Aleph[1] and Progol (Muggleton, 1995) are examples of classical ILP covering algorithms. The Progol algorithm was first implemented as C-Progol (a program written in C). Aleph is a later ILP system which implements Progol among other algorithms, depending on the settings chosen by the user.

Both Aleph and C-Progol work in the style of the covering algorithm AQ, first described in Michalski (1969). One positive example is removed at random from the dataset and the most specific conjunction of literals (hypothesis) which entails this example given the language restrictions is generated. This hypothesis is then repeatedly generalised, each time selecting a generalisation which covers as many positive examples as possible and as few negative examples (depending on the level of "noise" allowed). When it cannot be further generalised, this hypothesis is added to the hypothesis set, and the positive examples it covers are removed from the database. The process starts again, selecting a new positive example from the database and generating an overly specific hypothesis from it. This is then generalised, and the process repeated, until all positive examples have been covered by some hypothesis.

This type of learning usually requires both positive and negative examples (though sometimes a closed-world-assumption can be used if there are no negative examples, and work has been done on positive-only learning (Muggleton, 1996; Bostrom, 1998)). If the language of allowed literals is large then this top-down approach can be slow, since hypotheses are constructed before testing on the data. However, if the data set is large and the language is small then this approach can be very useful.

1.6.2 TILDE

TILDE (Blockeel *et al.*, 1998; Blockeel & De Raedt, 1998) is a first order variant of a decision tree algorithm. Nodes in the tree are relations, or literals in the language of Datalog, which can contain variables. These nodes (and paths from root to leaf) are tests on the data in the sense of the subsumption operation - a test is true if that literal (or set of literals) subsumes the data example. TILDE inherits the efficiency of decision trees, as it partitions the data set into smaller and smaller pieces each time. However, like decision trees, some hypotheses could be missed by this divisive split of the data, and errors made by a bad decision early on will be propagated down the tree.

[1] http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph.html

1.6.3 WARMR

WARMR (Dehaspe & De Raedt, 1997) is a first order association rule mining algorithm based on APRIORI for mining association rules in multiple relations. The language of Datalog (Ullman, 1988) is used for representing the data and background data. A language bias can be declared in the form of modes and types. It is a levelwise algorithm. However, the famous APRIORI_GEN function (see section 1.5.3) for generating candidates cannot be used directly, and instead, other methods of efficient candidate generation are employed. This is due to the fact that because of the language bias there may be predicate combinations which must happen in a certain order, and some predicates may not be used immediately.

In functional genomics there are many different data sources which can be used to build the databases to be mined, and so multiple relations are a convenient representation for the data. WARMR can be used as a data mining preprocessing step for genomic data, such as in King *et al.* (2000b), to extract important relationships.

1.7 Other machine learning methods

Many other machine learning methods exist including the following:

Neural networks Neural networks are learners which are based on the model of neurons in the brain (Minsky & Papert, 1969; Rumelhart & McClelland, 1986). They are interconnected networks of "neurons". A single neuron in a neural network is a simple weighted sum of its inputs, which is put through a thresholding function. The output of a neuron depends on whether the weighted sum exceeds the threshold or not. Neurons are often arranged in layers, with the inputs to one layer being the outputs from the previous layer. Neural networks are most commonly used as supervised learners which can learn complex classification boundaries if enough layers and neurons are used, and work best with numerical rather than symbolic data. They can be very accurate, but the intuition for the results may be very difficult to interpret. Mitchell (1997) gives a good description of the history and current work in neural networks.

Support vector machines are learners which can do binary classification and regression, usually with real valued attributes. They map non-linearly their n-dimensional input space into a high dimensional feature space. A linear classifier (separating hyperplane) is constructed in this high dimensional space to classify the data, and the hyperplane will be chosen to have the greatest distance (margin) between positive and negative examples. They are more suitable for problems with numerical data than symbolic data,

and are normally binary rather than multi-class classifiers. Outliers in the data can affect the position of the separating hyperplane, so data should be cleaned beforehand. They can be highly accurate, but as with neural networks, they are something of a black box when it comes to interpreting results (although the "support vectors" or data points closest to the hyperplane can be used as indicators of why the plane is where it is). Many books have been published on SVM theory and applications, including Cristianini (2000), and Vapnik (1998).

Genetic/Evolutionary algorithms borrow their inspiration from the process of evolution by natural selection found in nature (Fogel, 1995; Eiben, 2002; Spears *et al.*, 1993). They start with a population of possible hypotheses, and evaluate them on some training data. The best hypotheses are kept and used to create a new generation of hypotheses. New hypotheses can be obtained from old by two different operations: crossover and mutation. Crossover uses two hypotheses as "parents", randomly swapping over subsections of one hypothesis with subsections of the other, in the hope of combining the best of both. Mutation is where one hypothesis is deliberately altered in a small and random way, to introduce a change which could be beneficial but could also be detrimental. The new generation of hypotheses are evaluated and the best are used to produce the next generation. The whole process is iterated until the hypotheses are good enough to satisfy some criterion. This process can be slow and computationally expensive due to the slightly random nature of the search. The parameters which adjust mutation and crossover rate, and the fitness function which decides which hypotheses are the best and should be bred from, need to be carefully chosen to allow the algorithm to converge to good solutions without becoming trapped in certain areas of the search space.

Naive Bayes is a statistically based machine learning algorithm. It is based upon the direct application of Bayes Theorem and works under the assumption that the attributes are statistically independent from each other. It is used as a classifier on attribute-value data. It is one of the simplest machine learning algorithms available, and quick to use. Mitchell (1997) describes the use of Bayesian methods in machine learning.

Higher order learning These algorithms allow for even greater expressiveness than the first order algorithms. This creates a much greater search space. Kennedy and Giraud-Carrier (1999) uses evolutionary programming to try to tackle this, and uses a strongly typed language to help to restrict the search.

1.8 Evaluation of machine learning and data mining

Any results from machine learning must be evaluated before we can have any confidence in their predictions. There are several standard methods for evaluation.

1.8.1 Independent test data

Performance cannot be measured on the data used to train the classifier. This would give an overly optimistic measure of its accuracy. The classifier may overfit the training data and therefore evaluation on an independent test data set is the most common way to obtain an estimate of the classification accuracy on future unseen data. If the amount of data available is large then partitioning the data into training and independent test sets is the most common and fastest method of evaluation.

1.8.2 Cross validation

On smaller amounts of data, holding out a large enough independent test set may mean that not enough data is available for training. In this situation cross validation is preferred. The data is partitioned into a a fixed number (N) of partitions or "folds". N is commonly 10, although 3 is also popular. Each fold is held out as test data in turn, while the other $N - 1$ are used as training data. Performance of the classifiers produced for each of the N folds is measured and then the N estimates are averaged to give a final accuracy. Leave-one-out cross validation is standard cross validation taken to its extreme: instead of 10 folds, each individual datum is held out in turn so there are as many folds as items in the data set. This increases the amount of data available for training each time, but it is a computationally expensive process to build so many classifiers.

1.8.3 Bootstrap

The bootstrap is a method that constructs a training set by sampling **with replacement** from the whole data set (Efron & Tibshirani, 1993). The test set comprises the data that are not used in the training set. This means that the two sets are independent, but the training set can contain repeated items. This allows a reasonable sized training set to be chosen, while still keeping a test set. Kohavi (1995) compares the bootstrap and cross validation, show examples where each fails to give a good estimate of accuracy, and compares their results on standard data sets.

1.8.4 Validation data sets

Sometimes we need three independent data sets: a training set, a test set, and a third set (usually known as the validation set), that can be used to optimise parameters or select particular parts of a classifier. The validation set forms part of the procedure of creating the classifier and cannot be used in the final estimation of accuracy. We will usually use three independent data sets in our work.

1.8.5 Evaluating association mining

Evaluation of association mining algorithms is completely different. Each association mining algorithm should find exactly the same set of associations: all those above the minimum support value. Therefore the main difference between association mining algorithms is their efficiency (Freitas, 2000). They have no concern with overfitting or underfitting the data. Recently, extensions to association mining have explored measures of interestingness (Jaroszewicz & Simovici, 2001; Sahar, 1999; Padmanabhan & Tuzhilin, 1999) or generalisations and alternatives to the support measure (Liu *et al.*, 1999).

More information on methods of evaluation can be found in Witten and Frank (1999) for the standard methods, Cleary *et al* (1996) for use of the Minimum Description Length philosophy, and Lavrac *et al.* (1999) for a description of contingency tables and several different measures of rule accuracy.

1.9 Summary

This chapter presented the basic concepts in machine learning that will be used in this thesis. Decision trees, clustering, association rule mining and ILP were described in more detail, as these will be the main techniques used. In the next chapter we present the field of functional genomics, which will be the application area for the machine learning and data mining methods.

Chapter 2

An overview of functional genomics

2.1 Functional genomic terms

The determination of gene function from genomic information is what is known as functional genomics.

The central dogma of biology is that DNA is transcribed into RNA and RNA is translated into proteins. Figure 2.1 shows the relationship between the three. When we speak of gene function we usually mean the function of the products of genes after transcription and translation, which are proteins.

Proteins

Proteins are the molecules which do almost all the work in the cell. They are extremely important molecules, involved in everything from immunity to muscle structure, transportation, hormones, metabolism, respiration, repair, and control of genes. Understanding the roles of proteins is the key to understanding how the whole cell operates.

Proteins are polymers consisting of chains of amino acids. There are 20 different amino acids, so proteins can be represented by strings of characters for computational purposes. The structure and shape of the protein molecule (how the long chain of amino acids folds in 3-dimensional space) is relevant to the job of the protein. Much work has been done on protein structure determination, as it gives clues to the protein's function.

Protein structure can be described at various levels. The primary structure is the amino acid sequence itself. The secondary structure and tertiary structure describe how the backbone of the protein is arranged in 3-dimensional space. The backbone of the protein makes hydrogen bonds with itself, causing it to fold up

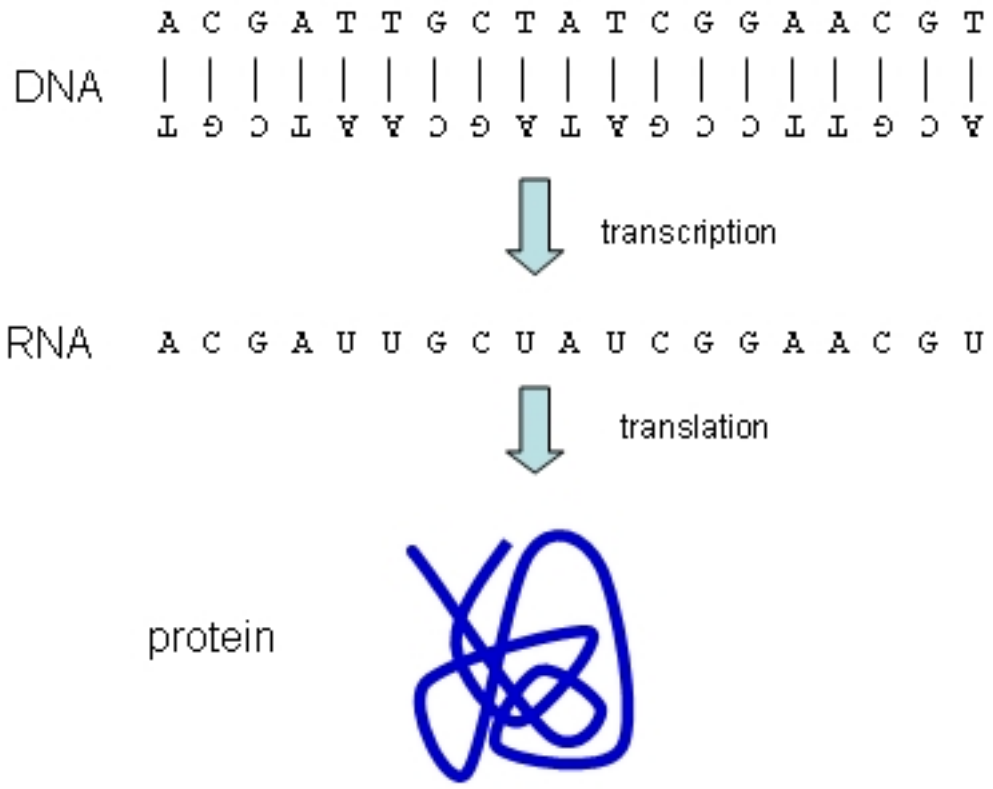
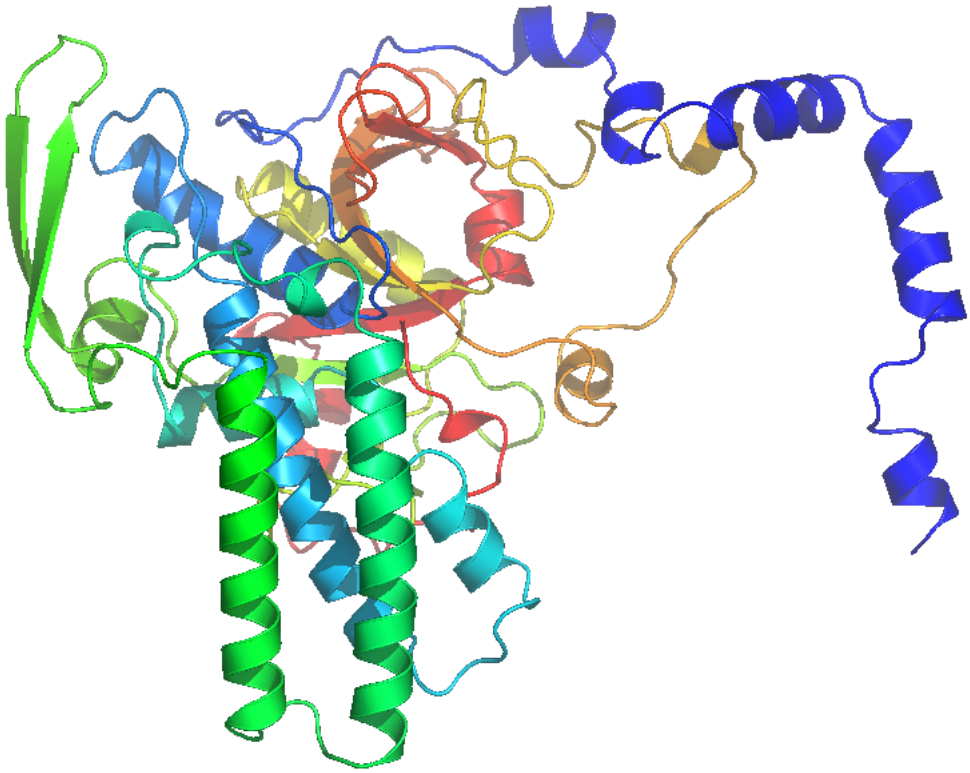


Figure 2.1: The central dogma of biology: information flows from DNA to RNA to proteins.



```
>1A6R:_ GAL6
MHHHHHHASENLAFQGAMASSIDISKINSWNKEFQSDLTHQLATTVLKNYNADDALLNKT
RLQKQDNRVFNTVVSTDSTPVTNQKSSGRAWLFAATNQLRLNVLSELNLKEFELSQAYLF
FYDKLEKANYFLDQIVSSADQDIDSRLVQYLLAAPTEDGGQYSMFLNLVKKYGLIPKDLY
GDLPYSTTASRKWNSLLTTKLREFAETLRTALKERSADDSIIVTLREQMQREIFRLMSLF
MDIPPVQPNEQFTWEYVDKDKKIHTIKSTPLEFASKYAKLDPSTPVSLINDPRHPYGKLI
KIDRLGNVLGGDAVIYLNVDNETLSKLVVKRLQNNKAVFFGSHTPKFMDKKTGVMDIELW
NYPAIGYNLPQQKASRIRYHESLMTHAMLITGCHVDETSKLPLRYRVENSWGKDSGKDGL
YVMTQKYFEEYCFQIVVDINELPKELASKFTSGKEEPIVLPIWDPMGALAK
```

Figure 2.2: A protein (yeast bleomycin hydrolase - PDB identification 1A6R). The 3 dimensional structure (secondary and tertiary structure) is shown in the image and the primary structure (amino acid sequence) is given as text. Alpha helices are the helix-like elements, and a beta sheet can be seen on the left of the molecule, represented by broad arrows.

into arrangements known as alpha helices, beta sheets and random coils. Alpha helices are formed when the backbone twists into right-handed helices. Beta sheets are formed when the backbone folds back on itself to make pleats. Random coils are neither random, nor coils, but are connecting loops that join together the alpha and beta regions. The alpha, beta and coil components are what is known as secondary structure. The secondary structures then fold up to give a tertiary structure to the protein. This makes the protein compact and globular. An example of the 3 levels of structure of a protein is given in Figure 2.2.

Other properties of proteins are also useful when determining function. Areas of hydrophobicity and polarity determine the shape of a protein and sites of interaction. The sequence length and molecular weight, and even just the ratios of the various amino acids have a bearing on the function of the protein. Sharing common patterns with other protein sequences, or common domains, can mean that the proteins have related function or evolved from a common ancestor. Evolutionary history or phylogeny of a protein can be used to understand why a protein was necessary and what its possible roles used to be.

Genes and ORFs

Genes are the units of heredity. They are sections of DNA which encode the information needed to make an organism and determine the attributes of that organism. Gene-finding programs are used to hypothesise where the genes lie in a DNA sequence. When an appropriate stretch of DNA (reasonable length, starting and ending with the right parts, etc.) is found, it is labelled as an Open Reading Frame or ORF - a putative gene. Most of the work in this thesis will use the word ORF instead of gene, as it is uncertain at this stage whether or not the sequences investigated are all real genes.

DNA

DNA is the molecular code of cells. It is a long chain molecule, consisting of a backbone of alternate sugar and phosphate groups, with a base attached to each sugar. There are 4 different bases which can be attached, and the sequence of the bases along the backbone makes the code. The bases are Adenine (A), Guanine (G), Cytosine (C), and Thymine (T). From a computer science perspective we would normally be dealing with DNA as a long string made up of the 4 letters A, G, C and T. The main purpose of DNA is to encode and replicate the information needed to make proteins.

The 4 bases of DNA are used in different combinations to code for the all the 20 amino acids that make proteins. A triplet of DNA bases is used to code for each amino acid. Figure 2.3 gives an example of this coding. As $4^3 = 64$, not 20, there is some redundancy in this coding, and there are several different ways to

code for some amino acids (though when there are several ways they tend to be closely related). Each triple of DNA is known as a *codon*. Apart from the codons which are used for amino acids, three of the triples are used to encode "stop" codons, which tell the cellular machinery where to stop reading the code.

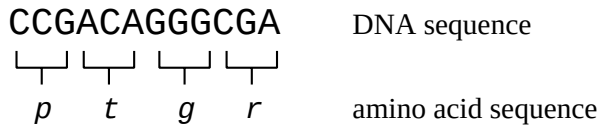


Figure 2.3: The DNA sequence is translated into a sequence of amino acids. Three DNA bases translate to one amino acid.

DNA is double stranded. It exists as two long chain molecules entwined together in the famous double helix. The two strands have complementary base pairing, so each C in one strand is paired with a G in the other and each A with a T. So when the size of DNA is quoted, it is usually in "base pairs". To give some idea of the size of the data: the DNA in the human genome is approximately $3 * 10^9$ base pairs (International human genome sequencing consortium, 2001), in the yeast genome *S. cerevisiae* it is approximately $13 * 10^6$ base pairs (Goffeau *et al.*, 1996), and in the bacterium *M. tuberculosis* it is approximately $4 * 10^6$ base pairs (Cole *et al.*, 1998).

Not all DNA codes for proteins. In mammals only about 5-10% does so. This percentage is much higher in bacteria (e.g. 90% coding in *M. tuberculosis*, 50-60% coding in *M. leprae*). The reason for the large amount of non-coding DNA is somewhat unclear, but it includes promoter and regulatory elements, highly repetitive DNA, and so-called "junk" DNA. There are theories which suggest some "junk" is for padding, so that the DNA is folded up in the correct position, and others which say it is the remnants of DNA which used to be coding, but has now become defunct or miscopied.

RNA

DNA is translated to proteins via RNA. RNA is a nucleic acid, very similar to DNA but single stranded, and the 4 bases of RNA are A, G, C and U (Uracil replaces Thymine). RNA is used for several roles in the cell. Its primary role is to take a copy of one of the strands of DNA. This piece of RNA (known as messenger RNA) might then undergo splicing to remove *introns*, pieces of sequence which

are non-coding, which interrupt the coding regions (*exons*) of a gene. Finally, the sequence of bases of RNA are then translated into amino acids to make the protein. Measurement of the RNA being produced ("expressed") in a cell can be used to infer which proteins are being produced.

Gene function

Even after a genome is fully sequenced, and the ORFs (or putative genes) have been located, we typically still do not know what many of them do. At the current time, approximately 40% of yeast ORFs have unknown function, and this figure is normal - in fact yeast is one of the best studied organisms. The functions of genes are usually determined either by sequence similarity to already known sequences, or by "wet" biology.

2.2 Functional genomics by biology

Previously biologists would work on discovering the function of just a few genes of interest, but recently there has been an increase in work on a genome-wide scale. For example, now there are genome wide knockout experiments where the genes are disrupted or "knocked out" and the organism grown under different conditions to see what effect the gene has if it is missing (Ross-Macdonald, 1999). And there are experiments to look at the genome wide "expression" of cells, that is, analysis of which RNA is currently being produced in the cell. Expression data can then be used to infer which genes are switched on under different environmental conditions, and hence the biological role of the genes. Ways to measure the expression of a cell include Northern blot analysis and SAGE. More recently, experiments are being done on a genome-wide scale with microarrays, a technique which can take a sample of the production of RNA in the cell at a point in time (DeRisi *et al.*, 1997; Eisen *et al.*, 1998; Zhang, 1999). Microarray technology has grown extremely popular and standard microarrays are being mass produced and widely used.

Winzeler and Davis (1997) describes various of the biological methods for functional genomics that have been applied to yeast. This includes expression analysis, proteomics and large-scale deletion and mutational analysis. Oliver *et al.* (1998) surveys a similar collection of techniques, with the added inclusion of metabolomic analysis. Table 2.1 summarises a selection of biological techniques for functional genomics mentioned in these surveys.

Functional genomics is currently a major area of research, as can be seen for example by the recent special supplement to Nature magazine, an "Insight" section devoted to functional genomics (Nature, 15th June 2000, 405(6788)).

Technology	Data	Description	Example references
gene disruption	phenotype	Disruption, mutation and deletion of genes. Comparison of the phenotype of the mutated organism with the original.	(Kumar *et al.*, 2000; Oliver, 1996)
microarray	transcriptome	Tiny chips spotted with complementary single stranded DNA. Measures expression of a cell by reading fluorescence levels of tagged RNA which hybridises to probes on chip.	(Eisen *et al.*, 1998; Gasch *et al.*, 2000)
SAGE	transcriptome	Serial Analysis of Gene Expression. Measures expression levels by using enzymatic reactions and sequencing to collect tags corresponding to mRNA molecules	(Velculescu *et al.*, 1997)
Northern blot analysis	transcriptome	Measurement of expression levels though gel electrophoresis	(Richard *et al.*, 1997)
protein-protein interactions	proteome	Discovery of protein-protein interactions by methods such as the yeast two-hybrid system, where a reporter gene is activated if the two proteins of interest interact	(Fromont-Racine *et al.*, 1997)
subcellular localisation	proteome	Discovery of the location of a protein within a cell, using immunofluorescence microscopy to observe staining patterns of tagged proteins	(Burns *et al.*, 1994)
metabolic footprinting	metabolome	The medium in which cells are grown is studied to observe the use of metabolites	(Raamsdonk *et al.*, 2001; Oliver *et al.*, 1998)

Table 2.1: Some biological techniques for functional genomics

2.3 Biological databases for functional genomics

There are many biological databases now available on the web, containing a wide variety of data. Some are dedicated specifically to proteins, such as SWISSPROT, which is a well-annotated, non-redundant, protein database, currently containing over 108,000 proteins. Each protein is annotated with pointers to published literature, keywords, comments, author, description and other fields. Other protein-related databases contain structural information, alignments, motifs and patterns, interactions, topology and crystallographic data.

There are also databases dedicated to DNA data (sometimes of one specific organism, sometimes of many). Many genomes have been completely sequenced now. Mostly these are either pathogens or model organisms. Among the pathogenic bacteria sequenced are those responsible for tuberculosis (*Mycobacterium tuberculosis*), diphtheria (*Corynebacterium diphtheriae*), whooping cough (*Bordetella parpertussis and Bordetella pertussis*), scarlet fever and toxic shock syndrome (*Streptococcus pyogenes*), typhoid fever (*Salmonella typhi*) and others. The Sanger Centre currently lists 24 pathogenic microbial genomes, and many more are on the way. Model organisms such as yeast (*Saccharomyces cerevisiae*), mouse (*Mus musculus*), fruit fly (*Drosophila melanogaster*), puffer fish (both *Fugu rubripes* and *Tetraodon nigroviridis*), Arabidopsis plant (*Arabidopsis thaliana*) and nematode worm (*Caenorhabditis elegans*) are chosen as standard laboratory testing organisms, usually fast and easy to grow, and representative of different classes.

There are databases for many other types of biological data, for example for holding results of microarray experiments, biomedical literature, phylogeny information, functional classification schemes, drug and compound descriptions and taxonomies. Some examples of currently popular databases are shown in Table 2.2. Every year in January the *Nucleic Acids Research* journal gives a review of databases available (Baxevanis, 2002).

These databases all contain errors which add noise to any computational methods. Errors can be as simple as a typographical mistake, or can be caused by experimental error, uncertainty or coincidence. Errors are often compounded by using using existing databases to infer new data (such as in sequence similarity search), without considering the source or reliability of the original data. Pennisi (1999) comments on the errors found in nucleotide sequences stored in GenBank, and the problems that these errors cause biologists who use GenBank as a reference point to find sequences similar to their gene of interest. She discusses methods currently used to try to reduce errors in DNA data, such as using automated screening for contaminants from commercial cloning kits, and incentives for people to correct faulty sequences or put effort into intensive database curation. Brenner (1999) investigated the scale of errors in genome annotation by comparing annotations from 3 different groups for the same genome (*M. genitalium*). He found an error rate of at least 8%; a result he terms “disappointing”.

Name	Description	Size (as of 23/10/02)
SWISS-PROT	Annotated protein sequences	116,269 entries
InterPro	Integrated Resources of Proteins Domains and Functional Sites	5,875 entries
PROSITE	Database of protein families and domains	1,574 different patterns, rules and profiles
PDB	Protein 3-D structure	19,006 structures
SCOP	Classification of protein structure	15979 PDB Entries, 39,893 Domains
EMBL	Nucleotide sequence database	23 billion nucleotides, 18 million entries
GenBank	Nucleotide sequence database	23 billion nucleotides, 18 million entries
DDBJ	DNA Data Bank of Japan	23 billion nucleotides, 18 million entries
MGI and MGD	mouse genome informatics and database	31,722 genes + various other
FlyBase	The Drosophila genome	more than 24,500 genes + various other
WormBase	The *C. elegans* genome	20,931 sequences + various other
TAIR	The Arabidopsis Information Resource	37,363 genes + various other
MIPS CYGD	Comprehensive Yeast Genome Database	various
SGD	Saccharomyces Genome Database	various
GenProtEC	*E. coli* genome and proteome database	various
EcoCyc	Encyclopedia of *E. coli* genes and metabolism	various
HGMD	Human Gene Mutation Database	30641 mutations
ENZYME	Enzyme nomenclature database	3982 categories
KEGG	Kyoto Encyclopedia of Genes and Genomes (metabolic pathways)	various
SMD	Stanford Microarray Database	various
Tree Of Life	Phylogeny and biodiversity database	various
PubMed	MEDLINE abstracts (literature database)	12 million citations
TRIPLES	Yeast phenotype, localisation and expression data	various

Table 2.2: A sample of the many databases used in functional genomics

The databases all have their own idiosyncratic formats and interfaces, and varying amounts of documentation to describe their contents. They are often designed for the one-lab-one-gene usage where a web interface is provided allowing a user to query their gene of interest. Querying on a genome-wide scale can require automatic web download tools which repeatedly query the interface, and parsing scripts to extract the required information from the HTML response.

Efforts to define new standards for data formats and exchange are in progress, using technology such as XML, and various standards bodies such as MIAME (MGED Working Group, 2001).

2.4 Computational biology

Computational biology is a new inter-disciplinary field where computer science is applied to biology in order to model systems, extract information, understand processes, etc. Knowledge of biology is needed to specify the problem and interpret the results, and knowledge of computer science is needed to devise the methods of analysis. Some examples of computational techniques currently used in biology include:

- Prediction of location of genes and analysis of sequence variation and properties (Li, 1999; Alba *et al.*, 1999; Salzberg *et al.*, 1996; Kowalczuk *et al.*, 1999; Grosse *et al.*, 2000; Loewenstern *et al.*, 1995).
  Gene-finding (or ORF-finding) programs are commonly used, and may well disagree on how many genes are present in an organism. Many properties of DNA sequences can be analysed statistically, such as G+C content, size of Open Reading Frames, base composition and amino acid composition. Simple statistics can be used to derive more interesting information: for example Alba *et al.* show that some amino acids are overrepresented in certain classes of proteins. Probabilistic/Bayesian models have been used to predict cellular localisation sites of proteins (Horton & Nakai, 1996).

- Assembly of sequenced fragments of DNA (Myers *et al.*, 2000).
  This article describes how the overlapping pieces of DNA read by a shotgun approach can be assembled together by computer to obtain the full sequence. Problems include repetitive DNA, gaps and unsampled segments, and lack of information about the ordering of the pieces.

- Prediction of protein structure (Domingues *et al.*, 2000; Maggio & Ramnarayan, 2001; Simon *et al.*, 2001; Schonbrun *et al.*, 2002; Sternberg, 1996; Guermeur *et al.*, 1999; Rost & Sander, 1993; Muggleton *et al.*, 1992; Ouali & King, 2000; Ding & Dubchak, 2001).
  Protein secondary and tertiary structure is important as a clue to protein

function, but determining structure experimentally by X-ray crystallography or NMR is time consuming and difficult, and the percentage of sequenced proteins for which a structure has been determined so far remains small. Computational methods of prediction of structure are so important that annual competitions (CASP - Critical Assessment of techniques for protein Structure Prediction, and CAFASP - Critical Assessment of Fully Automated Structure Prediction) take place to encourage research in this field.

- Support vector machines have been used for many discriminative problems in computational biology. Example applications include: discrimination between benign and pathologic human immunoglobulin light chains (Zavaljevski *et al.*, 2002), detecting remote protein homologies (Jaakkola *et al.*, 2000).

- Automatic extraction of information published in literature (Fukuda *et al.*, 1998; Baker *et al.*, 1999; Thomas *et al.*, 2000; Humphreys *et al.*, 2000; Leroy & Chen, 2002; Pustejovsky *et al.*, 2002).
  The amount of biomedical knowledge stored as the text of journals is now larger and faster growing than the human genome sequence (Pustejovsky *et al.*, 2002). Christened "the bibliome", this is an important source of data to be mined. Medline currently contains over 10 million abstracts, with 40,000 new abstracts added each month. The extraction of this type of data requires expert domain knowledge to hand code grammars and lexicons, but is now achieving respectable results.

- Automatic annotation of genomes (Andrade *et al.*, 1999; Möller *et al.*, 1999; Eisenhaber & Bork, 1999; Zhou *et al.*, 2002).
  Tools to aid annotation of genome sequences can be fully automatic or partially assisting, but all are invaluable when so many genome sequences must be annotated. Environments which help the annotator by offering additional information, links to online databases and sequence analysis tools allow more accurate and detailed annotations.

- Production of phylogenies and deduction of evolutionary history (Csuros, 2001; Wang *et al.*, 2002; Page & Cotton, 2002).
  Understanding the evolutionary history of an organism gives information about why certain features are found, possible gene function and ultimately how all organisms are related. Building and evaluating the many possible phylogenetic tree structures has been greatly aided by computer programs.

- Modelling of biological systems (Tomita, 2001; King *et al.*, 2000c; Reiser *et al.*, 2001).
  Whole system modelling and simulation of a whole cell is a grand aim. Parts

of the cell do not work in isolation, so a long term goal must be complete system modelling.

- Analysis of protein-protein interactions, genetic networks and metabolic pathways (Liang *et al.*, 1998; Koza *et al.*, 2001; D'Haeseleer *et al.*, 2000; Kurhekar *et al.*, 2002; Vu & Vohradsky, 2002; Bock & Gough, 2001).
  Specific interactions and pathways are part of the goal of understanding the whole cell. Modelling networks is mostly at the Bayesian level at the moment but computational graph theory also plays a part in their analysis.

- Mutagenicity and carcinogenicity can be predicted to some extent.
  This can be used to reduce the need for batteries of tests on animals. Inductive logic programming has been used to predict mutagenicity (using structure) (King *et al.*, 1996) and carcinogenicity in rodent bioassays (King & Srinivasan, 1996). Evolutionary rule learning has been used to predict carcinogenesis from structural information about the molecules (atoms and bond connectives) (Kennedy *et al.*, 1999).

- Discrimination of plant genotypes (Taylor *et al.*, 2002).
  Metabolomic data and neural networks were used to distinguish between two different Arabidopsis genotypes and also between the two possible genotypes of offspring produced by cross-breeding.

- Large scale analyses on distributed hardware (Waugh *et al.*, 2001).
  The volume of information is now so great that to be able to process it within a reasonable timescale, distributed and parallel processing bioinformatics infrastructures are beginning to be considered.

- Functional genomics (Eisen *et al.*, 1998; Koonin *et al.*, 1998; Marcotte *et al.*, 1999; Bork *et al.*, 1998; Gene Ontology Consortium, 2000).
  There has been a wide variety of work in this area, as understanding gene function is a holy grail of biology.

The last item in this list, "functional genomics" is of great importance, and we concentrate on this application in the rest of this thesis.

2.5 Functional genomics by computational methods

As the rate of genome sequencing grows steadily, and the amount of available data increases, computational functional genomics becomes both possible and necessary. Computational predictions can make experimental determination easier. Already, the first step in function prediction for a new gene is usually to do a sequence comparison against a database of known sequences (Shah & Hunter, 1997).

Unfortunately, this is sometimes as far as the determination of function goes, and many genes are left annotated as "putative" or "potential" without any indication of where that information came from. If the original sequence was annotated wrongly, then this error may be propagated through the databases though future gene annotation by sequence comparisons.

Bork *et al.* (1998) review the current process of computational prediction of function from sequence, in all the different stages, from studies of nucleotide frequencies and gene-finding, to proteomics and interdependencies of genes. Rastan and Beeley (1997) provide a similar review and discuss the use of model organism databases for functional genomics. They comment that computational and biological methods can complement each other:

> "The judicious melding of silicon-based and carbon-based biological research is set to take us forward faster than was imaginable a decade or so ago. The volume of genome data is driving the technology."

The following list gives some idea of the range of recent work in computational functional genomics.

- Improved sequence similarity search algorithms (Park *et al.*, 1997; Karwath & King, 2002; Pawlowski *et al.*, 2000; Williams & Zobel, 2002; Jaakkola *et al.*, 2000)
  Machine learning, intermediate sequences, better indexing schemes and new understanding of the relationship between sequence similarity and function can all be used to improve homology searches.

- Identification of motifs, alignments, protein fingerprints, profiles and other sequence based features that can be used to infer function (Hudak & McClure, 1999; Higgins *et al.*, 1994; Attwood *et al.*, 2002)
  Conserved regions of amino acids (motifs), multiple alignments, protein fingerprints and profiles can all be used to characterise protein families, which are likely to share common function.

- Sequence comparison to clusters of orthologous proteins can indicate function (Koonin *et al.*, 1998; Tatusov *et al.*, 2001).
  Orthologous genes are homologous genes that have diverged from each other over time as a consequence of speciation. Orthologous genes typically have the same function and so comparison to collections of othologs can indicate function.

- Microarray expression analysis (DeRisi *et al.*, 1997; Eisen *et al.*, 1998; Alon *et al.*, 1999; Heyer *et al.*, 1999; Törönen *et al.*, 1999; Tavazoie *et al.*, 1999; Butte & Kohane, 2000).

One of the most popular methods of functional genomics. Analysis of expression data can be used to infer similar functions for genes which show similar expression patterns. Most expression analysis uses unsupervised clustering, but other methods have also been tried. Supervised learning by support vector machines has been used (Brown *et al.*, 2000) to predict gene function. Rough sets have also been used to predict gene function from human expression data using the GeneOntology classification (Hvidsten *et al.*, 2001) and the Rosetta toolkit. Rosetta generates if-then rules using rough set theory, and has been used in several medical applications (Komorowski & Øhrn, 1999; Komorowski & Øhrn, 1999).

- Computational prediction of protein secondary structure (CASP, 2001; Wilson *et al.*, 2000; Ouali & King, 2000)
  Structure is used as an intermediate step to predicting the function (the 3-dimensional structure and shape of a protein is very pertinent to its function).

- Differential genome analysis (Cole, 1998)
  Comparing one genome with another can highlight areas where genes are not shared, and give a clue to their function, e.g. when comparing pathogenic and benign bacteria, or bacteria which are closely related in evolutionary terms.

- Combined approaches (Marcotte *et al.*, 1999; Hanisch *et al.*, 2002)
  Many approaches to functional genomics can now make use of several sources of data, including protein interaction data and expression data. Kretschmann *et al.* (2001) used C4.5 from the Weka package to predict SWISSPROT "keyword" field for proteins, given data about the taxonomy, INTERPRO classification, and PFAM and PROSITE patterns.

- Naive Bayes, C4.5 and Instance Based Learning were used to predict enzyme classification from sequence (des Jardins *et al.*, 1997).

- Work on ontologies and schemes for defining and representing function (Gene Ontology Consortium, 2000; Riley, 1998; Rubin *et al.*, 2002) is making progress towards standardising and understanding function.

Along with the use of computers comes a need to make terms and definitions rigorous and well defined. If we are to use computing to determine function then the concept of function must be defined first.

2.6 Functional annotation schemes

Biologists have always devised classification schemes, systematics, taxonomies and ontologies. These schemes help to organise and summarise information and provide a framework for understanding future inference of and historical reasons for properties.

Several classification schemes for gene function exist. Most schemes are specific to a particular genome, as in the beginning, few genomes had been sequenced and understood to any level of detail, and while it was important that the gene functions be classified for work on that genome, comparison to functions from other genomes was less of an issue. The functional classification scheme for genes from the *E. coli* bacterium was the first in 1993, and was produced 4 years before the genome itself was completely sequenced (Riley, 1993; Riley, 1998).

The Enzyme Commission (EC) system (Webb, 1992) is a scheme for enzyme classification. It classifies reactions in a 4 level hierarchy. However, it does not classify the properties of the enzymes involved in the reaction, nor the mechanism of the reaction, instead it classifies the changes that occur and the products of the reaction. It is also limited by assuming genes, enzymes and reactions have a one-to-one relationship, which is not generally true.

As more genomes were sequenced, more gene function classification schemes were produced. For example, the *M. tuberculosis* bacterium has a four level hierarchy for function of its ORFs produced by the Sanger Centre[1]. At the top (most general) level of the hierarchy there are the following classes:

- Small-molecule metabolism
- Macromolecule metabolism
- Cell Processes
- Other
- Conserved Hypotheticals
- Unknown

Each of these classes (except conserved hypotheticals and unknowns) are subdivided into more specific functions. Under "small molecule metabolism" there are classes such as "degradation" and "energy metabolism". These classes can then be further subdivided, and subdivided again, making a hierarchy up to 4 levels deep. An example of a classified ORF is Rv1905c (gene name AAO, description "D-amino acid oxidase"), which is classified as "small molecule metabolism → degradation → amino acids and amines"

[1] http://www.sanger.ac.uk/Projects/M_tuberculosis/

The previously mentioned classification scheme for *E. coli* (GenProtEC[2]) is similar to this hierarchy for *M. tuberculosis* in that it also assumes only one class per ORF, and is a strict hierarchy, though this time with only three levels. The top level classes in this hierarchy are the following:

- metabolism
- cryptic genes
- information transfer
- regulation
- transport
- cell processes
- cell structure
- location of gene products
- extrachromosomal
- DNA sites
- unknown

In this thesis, most of our work will be centred on the genome of the yeast *Saccharomyces cerevisiae*, which has several annotation schemes. MIPS (The Munich Information Center for Protein Sequences) gives a hierarchical classification[3] (Mewes *et al.*, 1999). This differs from the previously described classifications in that it allows ORFs to belong to more than one class, and sometimes they can belong as many as 10 classes. This is biologically realistic. However, it causes interesting problems for machine learning algorithms, where it is normally assumed that there is one class per data item (this shall be discussed in more detail later in Chapter 4). The Yeast Proteome Database (YPD) (Hodges *et al.*, 1999) classifies the function of yeast genes in 6 different dimensions, including genetic properties, functional category, and cellular role. However in each of these dimensions, the classification is broad and completely flat, not a hierarchy, but just a single level. The Saccharomyces Genome Database (SGD) in conjunction with the GeneOntology Consortium (GO) (Gene Ontology Consortium, 2000) produced another annotation scheme for yeast[4]. This is a scheme of three types of annotation:

[2]http://genprotec.mbl.edu/start

[3]http://mips.gsf.de/proj/yeast/CYGD/db/

[4]http://genome-www.stanford.edu/Saccharomyces/ and http://www.geneontology.org

molecular function, cellular component and biological process. As of October 26, 2002 GO contains 5,293 function, 1,118 component and 6,645 process terms. For the purpose of this work we shall be using just the molecular function annotations. The GO/SGD classification also differs from those mentioned before in that each type of annotation is a directed acyclic graph, rather than just a hierarchy. This means that any node in the graph may have more than one parent. For example, a "cell adhesion receptor" is a direct child of both "transmembrane receptor" and "cell adhesion molecule". The graph for molecular function is currently (as of 24/4/02) 11 levels deep in total.

GeneOntology is a classification scheme designed to cover many genomes. Current genomes for which GeneOntology annotations are available are the yeasts *S. cerevisiae* and *S. pombe*, the fruit fly, the plant *Arabidopsis thaliana*, the worm *C. elegans*, mouse and *V. cholerae*. This scheme is a major step forward to unifying results from multiple genomes and classifying gene function on a large scale. Its three-way annotation of molecular function, cellular component and biological process more accurately reflects the current understanding of the different types of function a gene can have (whereas other schemes usually mix the different types of function into a single list). And it allows many-to-many relationships between gene and function, rather than restricting each gene to a single functional class.

But even with all these different functional annotation schemes there is still a question about the suitability of the current functional classes for functional genomics. Kell & King (2000) discuss the arbitrariness of existing hand-generated classes which are based on our existing biological knowledge, what such classes should be used for, and how they should be decided upon. They conclude that "current lists of functional classes are not driven by data from whole-organism studies and are suboptimal for the purposes of functional genomics", and recommend that future classifications should be data driven, and that "inductive methods of machine learning provide the best initial approaches to assigning gene function".

2.7 Summary

This chapter presented the basic concepts in biology which will be used in this thesis. We surveyed the current state of the art in computational biology and in techniques and databases used in functional genomics. We also described functional annotation schemes that exist for classifying gene function. In the next chapter, we describe a method which has previously been successful in computational functional genomics and look at the problems we encounter when applying this method to the yeast genome.

Chapter 3

Initial work in function prediction

This chapter describes our initial work in predicting the functions of ORFs of unknown function. This used machine learning and data mining, and only used data that could be derived from ORF sequence. This work was published by King *et al.* (2000a; 2000c; 2001), and in the Ph.D. thesis of Andreas Karwath (Karwath, 2002).

The *M. tuberculosis* and *E. coli* genomes were used in this work because of the availability of data, and the importance of these bacteria. Table 3.1 shows a comparison of the *M. tuberculosis* and *E. coli* genomes.

Properties	*M. tuberculosis*	*E. coli*
Date sequenced	1998	1997
Genome size (million bp)	4.4	4.6
Number of genes	3924	4290
Genes of unknown function at time of experiment	1521 (39%)	942 (22%)

Table 3.1: *M. tuberculosis* and *E. coli* genome statistics

The aim was to learn accurate and understandable rules that could be used to predict gene function for these genomes.

3.1 *Mycobacterium tuberculosis*

M. tuberculosis is a Gram-positive bacterium that was sequenced in 1998 (Cole *et al.*, 1998). Currently, tuberculosis (TB) kills about 2 million people per year, and this year more people will die of TB than in any previous year. In 1993 the World Health Organisation declared TB a global emergency. TB is an infectious disease which is airborne and transmitted by coughs, sneezes and talking. Despite

the BCG vaccine and chemotherapy treatment, it continues to be a killer. The BCG vaccine has proven inefficient in several recent field trials (Andersen, 2001), and it is suspected that the bacterium used in the vaccine has lost many useful genes over the initial years of culture in the lab before better techniques for preserving live bacteria were invented (Behr *et al.*, 1999). Also, some *M. tuberculosis* strains are now multi drug resistant, and these are resistant to the two main drugs, isoniazid and rifampicin. The StopTB website* gives further information about TB, including campaigns, treatment methods and costs, and descriptions of the illness.

3.2 *Escherichia coli*

E. coli is a Gram-negative bacterium that was sequenced in 1997 (Blattner *et al.*, 1998) (strain K12). Humans carry *E. coli* bacteria in their intestines, and the large majority of these are normal and harmless. However, some strains can cause illnesses such as diarrhoea (particularly in children and travellers), urinary tract infections, gastroenteritis and neonatal meningitis. *E. coli* is commonly found on uncooked foods and in the environment. Cases of serious infection are often reported in the news, usually linked to food. One of the most virulent strains, O157:H7 has been sequenced this year, and is believed to have evolved within the last century. The sequence of the K12 strain is used in this work described in this chapter.

3.3 Data sets

For each organism (*M. tuberculosis* and *E. coli*) three types of data were collected. For each ORF in the organism this was:

SEQ: Data directly calculated from amino acid sequence, such as amino acid ratios and molecular weight

STR: Data calculated from secondary structure prediction

SIM: Data derived from similar sequences found by sequence similarity search of the SWISSPROT database

Dataset SEQ, directly calculated from amino acid sequence, consisted of simple attribute-value data. Approximately 430 attributes were calculated for each ORF, and most of these were numeric. Examples of attributes include amino acid ratios,

*http://www.stoptb.org

sequence length and molecular weight. King *et al.* (2000a) contains a detailed description of the attributes.

Datasets STR and SIM were relational data. This data was all expressed as Datalog, in large flat-file databases. STR facts describe location and relative positions of alpha helices, beta sheets and coils. SIM facts describe similar sequences that can be found in SWISSPROT, and their properties such as keywords, species and classification. Again, King *et al.* (2000a) contains a detailed description of this data.

The classes that were to be learned were taken from the Sanger Center functional classification scheme for *M. tuberculosis* and the GenProtEC scheme for *E. coli*, as described in section 2.6. These schemes are hierarchies (4 levels deep for *M. tuberculosis*, 3 levels deep for *E. coli*) and so we treated each level in the hierarchy separately.

3.4 Method

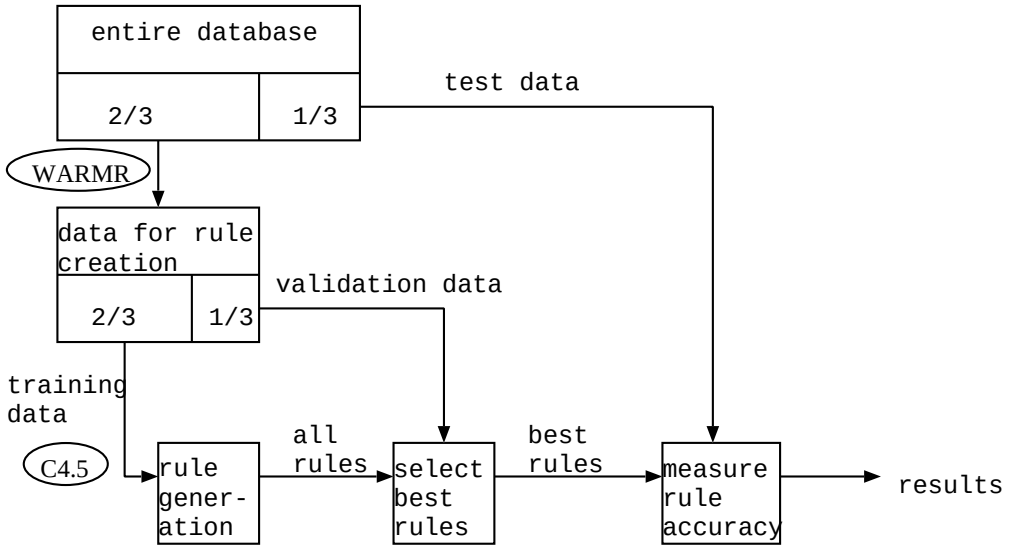


Figure 3.1: The data was split into 3 parts, training data, validation data and test data. Training data was used for rule generation, validation data for selecting the best rules and test data for measuring rule accuracy. All three parts were independent.

The method and flow of data in this work is shown in Figure 3.1. One third of the database was held out for testing the accuracy of the method. The rest of the data for datasets STR and SIM were mined using the WARMR algorithm to

find thousands of frequent patterns. WARMR is an Inductive Logic Programming algorithm that finds frequent associations (see section 1.5) in first order relational data such as this. It is well suited as a preprocessing step for relational data before input to a propositional learning algorithm. This is because the frequent first order patterns that it finds can be considered to be the interesting features in the dataset. These features can then be used in a propositional learning algorithm, reducing the complexity of the problem and allowing faster algorithms to be used. In this case, these frequent associations were then used as boolean attributes for each ORF - having a value of 1 if the association was present for that ORF, and 0 if not present.

The next step in the procedure was standard machine learning, using the well-known decision tree algorithms C4.5 and C5.0 on this attribute-value data to generate rules. As C4.5 and C5.0 are supervised learners, only the ORFs with known functions were used to generate the rules. Also, one third of this dataset was held out as a validation set, to estimate the generalisation error of the rules (to allow the best rules to be chosen), and this validation set was not used in rule generation. In this machine learning procedure the requirements were not for complete coverage of the data set, but for general and accurate rules that could be used for future prediction. This is an unusual machine learning problem (the usual setting is to find the best classifier that classifies the whole data set (Witten & Frank, 1999; Provost *et al.*, 1998)). Therefore the validation set was important, filtering out rules which were overfitting the training data.

The final stage was to apply the selected rules to the held out test data, to get a measure of the predictive accuracy of the rule set, and then to apply the rules to the data from the ORFs with unknown function, to make predictions for their function.

The classes that were to be predicted came from the Sanger Centre for *M. tuberculosis* and GenProtEC for *E. coli*, as described in Section 2.6. Both these classification schemes are hierarchies, 4 levels deep for *M. tuberculosis* and 3 levels deep for *E. coli*. Each level in the hierarchy was treated independently.

3.5 Results

Predictions were made for 24% of the unknown *E. coli* ORFs and 65% of the *M. tuberculosis* ORFs. The accuracy of the rules on the test sets were between 61 and 76%. Table 3.2 shows the accuracy, number of rules found and number of predictions made.

Examples of the rules that were discovered are given in Figures 3.2 and 3.3. Figure 3.2 shows a very simple rule for *M. tuberculosis*. This rule is 85% accurate on the test set and covers many proteins involved in protein translation. It is consistent with protein chemistry, as lysine is positively charged, which is desirable

	M. tuberculosis				*E. coli*		
	Level 1	Level 2	Level 3	Level 4	Level 1	Level 2	Level 3
Number of rules found	25	30	20	3	13	13	13
Average test accuracy	62%	65%	62%	76%	75%	69%	61%
Default test accuracy	48%	14%	6%	2%	40%	21%	6%
New functions assigned	886	507	60	19	353	267	135

Table 3.2: Results for *M. tuberculosis* and *E. coli*. The number of rules found are those selected on the validation data set. Average test accuracy is the accuracy of the predictions on the test proteins of assigned function (if conflicts occurred the prediction with the highest *a priori* probability was chosen). Default test accuracy is the accuracy that could be achieved by always selecting the most populous class. 'New functions assigned' is the number of ORFs of unassigned function predicted. Level 1 is the most general classification level in each case, with levels 2, 3 and 4 being progressively more specific.

if the percentage composition of lysine in the ORF is < 6.6
then its functional class is 'macromolecule metabolism'
Test set accuracy: 85%

Figure 3.2: A simple rule found for *M. tuberculosis*

for interaction with negatively charged RNA. Figure 3.3 shows a more complex rule. This rule is 80% accurate on the test set, and although it is non-intuitive, this accuracy cannot be explained by chance; it must therefore represent some real biological regularity.

For *M. tuberculosis* datasets SEQ and SIM only were used (secondary structure prediction was not computed) and the two datasets were simply merged before application of C4.5/C5.0. When the *E. coli* data was processed all three datasets were available. To allow comparison of the utility of the three different types of data in biological function prediction, combinations of the three datasets were investigated. Voting methods were implemented where the rules produced by each dataset were allowed to vote for the prediction of an ORF's class. The results were compared with the direct combination of different types of data prior to learning with C4.5/C5.0, and with results from the individual data sets themselves. Table 3.3 shows the accuracy of different datasets and combinations for *E. coli*.

The richer data in dataset SIM was found to give the best predictive power, as intuitively expected. However, the other two datasets still did remarkably well. This study compared simple combination of the different types of attributes

if the ORF's percentage composition of the dipeptide tyr-arg is ≤ 0.054
and no homologous protein was found annotated with the keyword ''alternative splicing''
and a homologous protein was found in *H. Sapiens*
and a homologous protein was found of low sequence similarity
and no homologous protein was found of very high sequence similarity and very low asn ratio
and a homologous bacterial protein was found with a very high molecular weight
and a homologous proteobacteria protein was found annotated with the keyword 'transmembrane' and a very high molecular weight
and no homologous protein was found in *E. coli* with very high leu percentage composition and normal molecular weight
then its functional class is 'small molecule metabolism, degradation, fatty acids'
Test set accuracy: 80%

Figure 3.3: A complex rule found for *M. tuberculosis*

before learning with various other methods of combination and discovered that simple voting strategies after separate training on each dataset performed best. A trade-off of coverage against accuracy is always unavoidable in machine learning, and this was no exception. By changing the combination method, rules could be selected which were highly accurate but covered fewer ORFs (from a voting system that only selected predictions made by at least two datasets), or less accurate but applied to many more ORFs (from a weighted voting system). The full results are reported in King *et al.* (2001).

This author's contribution to this work

This work was done jointly by Andreas Karwath, Ross D. King, Luc Dehaspe and Amanda Clare. This author's contribution to this work was in evaluation of results and construction and application of voting methods.

	level		
datasets	1	2	3
SEQ	64	63	41
SIM	75	74	69
STR	59	44	17
SEQ+SIM	84	71	60
SEQ+STR	69	64	50
SIM+STR	75	69	54
SEQ+SIM+STR	75	69	61
WTD_VOTE_ALL	60	54	42
VOTE_2_ALL	75	68	68
WTD_VOTE_SSS	64	66	52
VOTE_2_SSS	86	88	90

Table 3.3: Accuracy in percentages of the different datasets and combinations of datasets for *E. coli*. WTD_VOTE_ALL is a weighted vote from each ruleset where the accuracy on the validation set is used to weight the vote of a rule ("ALL" indicates that the votes could come from all of SEQ, SIM, STR, SEQ+SIM, SEQ+STR, SIM+STR and SEQ+SIM+STR). VOTE_2_ALL only uses predictions that are made by at least 2 of the rulesets. WTD_VOTE_SSS is a weighted vote from SEQ, SIM and STR only. VOTE_2_SSS is a vote from at least 2 of SEQ, SIM and STR.

3.6 Conclusion

These results were extremely promising. If function of ORFs can be predicted with such accuracy, this could greatly aid biologists in experimentally testing for ORF function.

Ideally, the results should be experimentally confirmed. This would be an important step in convincing biologists of the value of computational methods like these. If we were to make predictions for another organism that is easier to handle, such as the yeast *S. cerevisiae*, then the predictions are more likely to be experimentally tested. *S. cerevisiae* has a larger genome and has a much richer collection of data available, due to its early sequencing and it being relatively easy to manipulate.

3.7 Extending our methodology to yeast

3.7.1 *Saccharomyces cerevisiae*

Saccharomyces cerevisiae (baker's or brewer's yeast) is a model eukaryotic organism. Many genes in *S. cerevisiae* are similar to genes in mammals, including humans, and several genes are similar to genes which are linked to disease in humans. *S. cerevisiae* was the first eukaryotic genome sequence to be completed, in 1996 (Goffeau *et al.*, 1996). It has 16 chromosomes, consisting of 13.3 million base pairs which contain approximately 6,300 protein encoding ORFs. It is larger and more complex than *M. tuberculosis* or *E. coli* and its genome is compared to those of *M. tuberculosis* and *E. coli* in Table 3.4.

Properties	*S. cerevisiae*	*M. tuberculosis*	*E. coli*
Date sequenced	1996	1998	1997
Genome size (million bp)	13.3	4.4	4.6
Number of chromosomes	16	1	1
Classification scheme	MIPS	Sanger Centre	GenProtEC
Number of genes	6,300	3,924	4,290
Genes of unknown function at time of experiment	2514 (40%)	1521 (39%)	942 (22%)

Table 3.4: Comparison of *S. cerevisiae* genome to those of *M.tuberculosis* and *E. coli*

S. cerevisiae is cheap and quick to grow, non-pathogenic, and easy to manipulate genetically. It has been the focus of detailed study over the years. These studies include expression analysis via Northern blots (Richard *et al.*, 1997), SAGE (Velculescu *et al.*, 1997) and microarrays (DeRisi *et al.*, 1997; Cho *et al.*, 1998; Chu *et al.*, 1998; Spellman *et al.*, 1998; Gasch *et al.*, 2000), 2-d gel electrophoresis (Boucherie *et al.*, 1995), two-hybrid systems for protein-protein interactions (Fromont-Racine *et al.*, 1997), large scale deletion and mutational analysis (Ross-Macdonald, 1999; Kumar *et al.*, 2000; Oliver, 1996) and phenotypic analysis (Oliver, 1996).

Despite all this biological knowledge and investigation, approximately 40% of the ORFs in yeast remain without clear function or purpose.

3.7.2 Additional challenges for yeast

Working with yeast brings additional **machine learning challenges**.

- *M. tuberculosis* and *E. coli* were annotated with a single function per ORF, however this assumption is in general incorrect. The annotations provided

for yeast usually show several different functions for each ORF, with sometimes more than 10 different functional roles recorded for some ORFs. This means that we cannot use the same method - C4.5 and C5.0 expect a single class label for each data item. The problem of **multiple labels** is an interesting machine learning problem which has so far been little addressed in the field. This problem is discussed further in later sections of this thesis.

- Another interesting issue for machine learning is to make more use of the structure of the data we have. Functional classification schemes for ORFs are usually organised in a hierarchy, with general classes at the top level of the hierarchy, each being subdivided into more specific classes in the next level down, and each of these divided in turn. Each ORF is annotated as belonging to classes at various levels in the hierarchy. In the work on *M. tuberculosis* and *E. coli* these hierarchies were flattened, and machine learning was applied separately for each level of classes. This treats all classes within a level as independent, whereas in reality, some will be very similar to each other. The problem of **hierarchical classification** is also discussed further in later sections of this thesis. As well as hierarchical classes we also have **hierarchically structured data**, such as the taxonomy of the species a protein belongs to. Instead of simply flattening such a structure it would be beneficial to be able to use it.

- The recent increase in availability of expression data from microarray experiments highlights the problem of machine learning using short **time-series data** (Morik, 2000). This involves representation and algorithmic issues, since the time series from expression data are too short for standard methods to be useful. If we were to choose a propositional learner: the element of time would be difficult to represent, and may have to be ignored. If we were to choose a relational (ILP) learner: we could express the relations between different time steps, but the numerical nature of the data may cause problems, since ILP systems are generally designed for symbolic data. **Discretisation** (conversion of continuous data into symbolic data) can be done in various ways, each with their advantages and drawbacks.

- The **increased size of data** relative to *M. tuberculosis* and *E. coli* will severely test the machine learning algorithms and some may need to be adapted. Yeast has more ORFs (half as many again as the bacteria). In addition each yeast ORF has many more homologs than the ORFs from *M. tuberculosis* and *E. coli*. This is partly due to the size of the databases increasing each year so more potential homologs are now available. Also, the general frequency of eukaryotic ORFs in SWISSPROT is greater than that of ORFs from bacteria, so a eukaryote such as yeast is more likely to find similarities.

The following chapters explain these issues and an investigation into their solutions in more detail.

Chapter 4

Phenotype data, multiple labels and bootstrap resampling

The work in this chapter has been published as Clare & King (2001), Clare & King (2002b).

4.1 Determining gene function from phenotype

The phenotype of an organism is its observable, physical characteristics. Perhaps the least analysed form of genomics data is that from phenotype experiments (Oliver, 1996; Mewes *et al.*, 1999; Kumar *et al.*, 2000). In these experiments specific genes are removed from the cells to form mutant strains, and these mutant strains are grown under different conditions with the aim of finding growth conditions where the mutant and the wild type (no mutation) differ. This approach is analogous to removing components from a car and then attempting to drive the car under different conditions to diagnose the role of the missing component. Function of genes can be determined by removing the gene and observing the resulting effect on the organism's phenotype.

4.2 Phenotype data

Three separate sources of phenotypic data were available: TRIPLES (Kumar *et al.*, 2000), EUROFAN (Oliver, 1996) and MIPS (Mewes *et al.*, 1999).

- The TRIPLES (TRansposon-Insertion Phenotypes, Localization and Expression in *Saccharomyces*) data were generated by randomly inserting transposons into the yeast genome.
  URLs: http://ygac.med.yale.edu/triples/triples.htm, (raw data)
  http://bioinfo.mbb.yale.edu/genome/phenotypes/ (processed data)

- EUROFAN (European functional analysis network) is a large European network of research which has created a library of deletion mutants by using PCR-mediated gene replacement (replacing specific genes with a marker gene (kanMX)). We used data from EUROFAN 1.
  URL: http://mips.gsf.de/proj/eurofan/

- The Munich Information Center for Protein Sequences (MIPS) database contains a catalogue of yeast phenotype data.
  URL: http://mips.gsf.de/proj/yeast/

The data from the three sources were combined together to form a unified dataset, which can be seen at http://www.aber.ac.uk/compsci/Research/bio/dss/phenotype/. The phenotype data set has the form of attribute-value vectors: with the attributes being the growth media, the values of the attributes being the observed sensitivity or resistance of the mutant compared with the wildtype, and the class the functional class of the gene.

The values that the attributes could take are shown in Table 4.1.

n	no data
w	wild-type (no phenotypic effect)
s	sensitive (less growth than for the wild-type)
r	resistance (better growth than for the wild-type)

Table 4.1: Attribute values for the phenotype data

Notice that these data were not available for all genes due to some mutants being not viable or untested, and not all growth media were tested/recorded for every gene, so there were *very many missing values* in the data.

There were 69 attributes. 68 of these were the various growth media (e.g. calcofluor_white, caffeine, sorbitol, benomyl). The final attribute was the (discretised) number of media that that had caused a reaction in the mutant (i.e. for how many of the attributes the mutant had a value of either "s" or "r").

4.3 Functional class

Genes may have more than one functional class. This is reflected in the MIPS classification scheme for *S. cerevisiae* (where a single gene can belong to up to 10 different functional classes). This means that the classification problem is a *multi-label* one (as opposed to *multi-class* which usually refers to simply having more than two possible disjoint classes for the classifier to learn). Moreover, we are not looking for a classifier to give a range of possible/probable classes. The

multi-label case is where we wish to predict a conjunction of classes, rather than a disjunction. For example, the ORF YBR145W (alcohol dehydrogenase V) has roles in all of "C-compound and carbohydrate utilization", "fermentation" and "detoxification". Thus we are not looking for a classifier that determines class rankings or divides up probabilities of class membership, rather, one that predicts a set of classes.

There is only a limited literature on such problems, for example Karalič & Pirnat (1991), McCallum (1999), and Schapire & Singer (2000). Karalič & Pirnat (1991) used the obvious method of learning a separate classifier for each binary classification problem and combining the results. McCallum (1999) describes a Bayesian approach to multi-label learning for text documents. This uses class sets, selected from the power set of classes. The class sets are built up in size in a similar style to the levelwise approach of building associations in association mining algorithms. He evaluates his approach on documents in ten classes from a Reuters collection of documents. Schapire & Singer (2000) describes extensions of AdaBoost which were designed to handle multiple labels by using classification methods that produce a ranking of possible classes for each document, with the hope that the appropriate classes fall at the top of the ranking. They too tested on Reuters collections.

Other literature about multi-label problems follow Karalič & Pirnat's example and simply build multiple classifiers and combine the results. This is reasonable because the number of labels is usually small, chosen to be a subset of available classes with sufficient examples to demonstrate the effectiveness of a machine learning method. The UCI repository (Blake & Merz, 1998) currently contains just one dataset ("University") that can be considered a multi-label problem. (This dataset shows the academic emphasis of individual universities, which can be multi-valued, for example, business-education, engineering, accounting and fine-arts). Weka, one of the most popular free software machine learning environments, has no support for multi-label learning despite regular questions on the mailing list to ask whether this feature exists (for example see the threads: August 2002 "Support for multilabel categorization" and October 2002 "About multiclass").

The simplest approach to the multi-label problem is to learn separate binary classifiers for each class (with all genes not belonging to a specific class used as negative examples for that class). However this is clearly cumbersome and time-consuming when there are many classes - as is the case in the functional hierarchy for yeast. Also, in sparsely populated classes there would be very few positive examples of a class and overwhelmingly many negative examples. As our data are propositional and we are looking for a discrimination algorithm we chose to develop a new algorithm based on the successful decision tree algorithm C4.5 (Quinlan, 1993).

Our functional classification scheme is also a hierarchy. For the time being, we deal with the class hierarchy by learning separate classifiers for each level. This simple approach has the unfortunate side-effect of fragmenting the class structure and producing many classes with few members - e.g. there are 99 potential classes represented in the data for level 2 in the hierarchy. We therefore needed to develop a resampling method to deal with the problem of learning rules from sparse data and few examples per class.

A extra challenge in this work is to learn a set of rules which accurately predict functional class. This differs from the standard statistical and machine learning supervised learning task of maximising the prediction accuracy on the test set. Measures must be taken to ensure we do not overfit the data, and collect only general and accurate rules for future prediction.

4.4 Algorithm

The machine learning algorithm we chose to adapt was the well known decision tree algorithm C4.5 (Quinlan, 1993). C4.5 is known to be robust, and efficient (Elomaa, 1994). The output of C4.5 is a decision tree, or equivalently a set of symbolic rules (see Section 1.3 for an introduction to decision trees). The use of symbolic rules allows the output to be interpreted and compared with existing biological knowledge - this is not generally the case with other machine learning methods, such as neural networks, or support vector machines.

In C4.5 the tree is constructed top down. For each node the attribute is chosen which best classifies the remaining training examples. This is decided by considering the information gain, which is the difference between the entropy of the whole set of remaining training examples and the weighted sum of the entropy of the subsets caused by partitioning on the values of that attribute.

$$information\_gain(S, A) = entropy(S) - \sum_{v \in A} \frac{|S_v|}{|S|} * entropy(S_v) \tag{4.1}$$

where A is the attribute being considered, S is the set of training examples being considered, and S_v is the subset of S with value v for attribute A. The algorithms behind C4.5 are well documented and the code is open source, so this allowed the algorithm to be extended.

Multiple labels are a problem for C4.5, and almost all other learning methods, as they expect each example to be labeled as belonging to just one class. For yeast this is not the case, as a gene may belong to several different classes. In the case of a single class label for each example the entropy for a set of examples is just

$$entropy(S) = -\sum_{i=1}^{N} p(c_i) \log p(c_i) \tag{4.2}$$

where $p(c_i)$ is the probability (relative frequency) of class c_i in this set.

We need to modify this formula for multiple class labels. Entropy is a measure of the amount of uncertainty in the dataset. It can be thought of as follows: given an item of the dataset, how much information is needed to describe that item? This is equivalent to asking how many bits are needed to describe all the classes it belongs to.

To estimate this we sum the number of bits needed to describe membership or non-membership of each class. In the general case where there are N classes and membership of each class c_i has probability $p(c_i)$ the total number of bits needed to describe an example is given by

$$entropy(S) = -\sum_{i=1}^{N} ((p(c_i) \log p(c_i)) + (q(c_i) \log q(c_i))) \tag{4.3}$$

where
$p(c_i)$ = probability (relative frequency) of class c_i
$q(c_i) = 1 - p(c_i)$ = probability of not being member of class c_i

This formula is derived from the basic rule of expectation. We need to calculate the minimum number of bits necessary to describe all the classes an item belongs to. For a simple description, a bitstring could be used, 1 bit per class, to represent each example. With 4 classes {**a**,**b**,**c**,**d**}, an example belonging to classes **b** and **d** could be represented as 0101. However, this would usually be more bits than actually needed. Suppose every example was a member of class **b**. In this case the second bit would not be needed, as class **b** membership is assumed. Suppose instead that 75% of the examples were members of class **b**. Then it is known in advance that an example is more likely to belong to class **b** than not to belong. The expected amount of information gained by actually knowing whether or not it belongs will be:

$$\begin{aligned} p(\text{belongs}) * gain(\text{belongs}) + p(\text{does not belong}) * gain(\text{does not belong}) & \\ = \; & 0.75 * (\log 1 - \log 0.75) + 0.25 * (\log 1 - \log 0.25) \\ = \; & -(0.75 * \log 0.75) - (0.25 * \log 0.25) \\ = \; & 0.81 \end{aligned}$$

where $gain(x)$ = information gained by knowing x

That is, only 0.81 of a bit is needed to represent the extra information required to know membership or non-membership of class **b**. Generalising, we can say that instead of one bit per class, what is needed is the total of the extra information necessary to describe membership or non-membership of each class. This sum will be

$$-\sum_{i=1}^{N}((p(c_i)\log p(c_i)) + (q(c_i)\log q(c_i)))$$

where $p(c_i)$ is probability of membership of class c_i and $q(c_i)$ is probability of non-membership of class c_i.

Now the new information after a partition according to some attribute, can be calculated as a weighted sum of the entropy for each subset (calculated as above), where this time, weighted sum means if an item appears twice in a subset because it belongs to two classes then we count it twice.

In allowing multiple labels per example we have to allow leaves of the tree to potentially be a set of class labels, i.e. the outcome of a classification of an example can be a set of classes. When we label the decision tree this needs to be taken into account, and also when we prune the tree. When we come to generate rules from the decision tree, this can be done in the usual way, except when it is the case that a leaf is a set of classes, a separate rule will be generated for each class, prior to the rule-pruning part of the C4.5rules program (part of the C4.5 package). We could have generated rules which simply output a set of classes - it was an arbitrary choice to generate separate rules, chosen for comprehensibility of the results. Appendix A describes in more technical detail the various changes that were required to the code of C4.5.

4.5 Resampling

The large number of classes meant that many classes have quite small numbers of examples. We were also required only to learn a set of accurate rules, not a complete classification. This unusual feature of the data made it necessary for us to develop a sophisticated resampling approach to estimating rule accuracy based on the bootstrap.

All accuracy measurements were made using the m-estimate (Cestnik, 1990). This is a generalisation of the Laplace estimate, taking into account the *a priori* probability of the class. The m-estimate for rule r (M(r)) is:

$$M(r) = \frac{p + m\frac{P}{P+N}}{p + n + m}$$

where

P	=	total number of positive examples
N	=	total number of negative examples
p	=	number of positive examples covered by rule r
n	=	number of negative examples covered by rule r
m	=	parameter to be altered

Using this formula, the accuracy for rules with zero coverage will be the *a priori* probability of the class. m is a parameter which can be altered to weight the *a priori* probability. We used $m = 1$.

The data set in this case is relatively small. We have 2452 genes with some recorded phenotypes, of which 991 are classified by MIPS as "Unclassified" or "Classification not yet clear-cut". These genes of unknown classification cannot be used in supervised learning (though we can later make predictions for them). This leaves just 1461, each with many missing values. At the top level of the classification hierarchy (the most general classes), there are many examples for each class, but as we move to lower, more specific levels, the classes become more sparsely populated, and machine learning becomes difficult.

We split the data set into 3 parts: training data, validation data to select the best rules from (rules were chosen that had an accuracy of at least 50% and correctly covered at least 2 examples), and test data. We used the validation data to avoid overfitting rules to the data. However, splitting the dataset into 3 parts means that the amount of data available for training will be greatly reduced. Similarly only a small amount will be available for testing. Initial experiments showed that the split of the data substantially affected the rulesets produced, sometimes producing many good rules, and sometimes none. The two standard methods for estimating accuracy under the circumstance of a small data set are 10-fold cross-validation and the bootstrap method (Kohavi, 1995; Efron & Tibshirani, 1993). Because we are interested in the rules themselves, and not just the accuracy, we opted for the bootstrap method because a 10-fold cross validation would make just 10 rulesets, whereas bootstrap sampling can be used to create hundreds of samples of the data, and hence hundreds of rulesets. We can then examine these and see which rules occur regularly and are stable, not just artifacts of the split of the data.

The bootstrap is a method where data are repeatedly sampled with replacement to make hundreds of training sets. A classifier is constructed for each sample, and the accuracies of all the classifiers can be averaged to give a final measure of accuracy. First a bootstrap sample was taken from the original data. Items of the original data not used in the sample made up the test set. Then a new sample was taken with replacement *from this sample*. This second sample was used as training data, and items that were in the first sample but not in the second made up the validation set. All three data sets are non-overlapping. Figure 4.1 shows how the 3 data sets were composed from repeated sampling.

We measured accuracy on the held-out test set. We are aware that this will give a *pessimistic* measure of accuracy (i.e. the true accuracy on the whole data set will be higher), but this is acceptable.

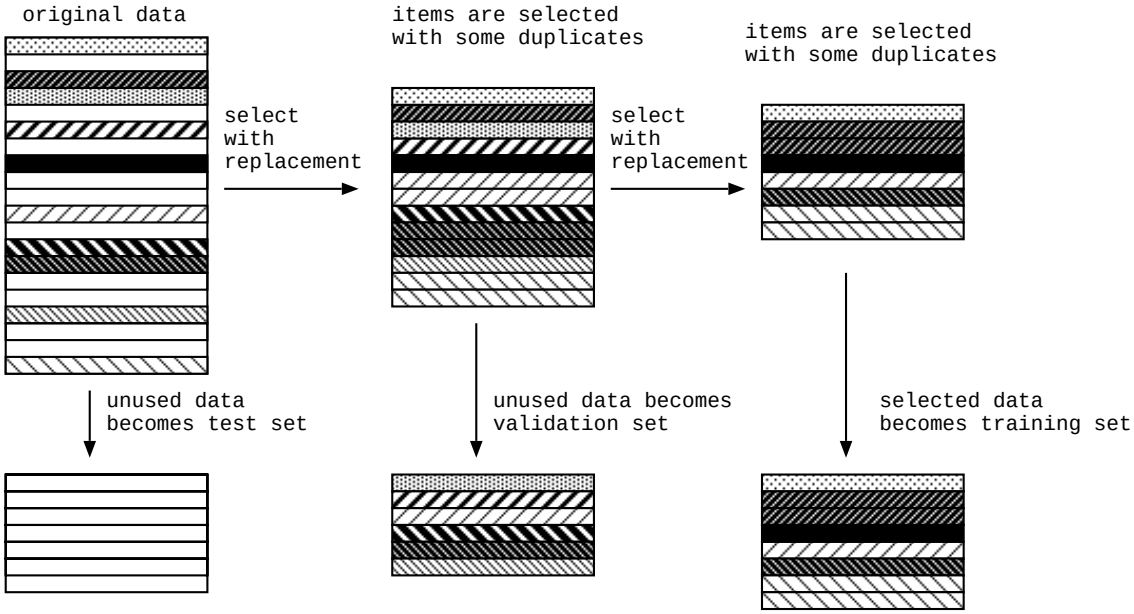


Figure 4.1: A bootstrap sample is taken. Items not used in this sample make up the test set. Then a sample is taken from the sample. Items not used this time make up the validation set, and the final sample becomes the training set.

4.6 Results

The classification scheme was the MIPS functional hierarchy*, using the catalogue as it was on 27 September 1999. 500 bootstrap samples were made, and so C4.5 was run 500 times and 500 rulesets were generated and tested. To discover which rules were stable and reliable we counted how many times each rule appeared across the 500 rulesets. Accurate stable rules were produced for many of the classes at levels 1 and 2 in the hierarchy. At levels 3 and 4 (the most specific levels with the least populated classes) no useful rules were found. That is, at the lower levels, few rules were produced and these were not especially general or accurate.

Table 4.2 shows some general statistics for the rulesets. Due to the nature of the bootstrap method of collecting rules, only *average* accuracy and coverage can be computed (rather than *total*), as the test data set changes with each bootstrap sample.

Table 4.3 shows the number of rules found for the classes at level 1. We did not expect to be able to learn rules for every class, as some classes may not be distinguishable given the growth media that were used.

Many biologically interesting rules were learned. The good rules are generally very simple, with just one or two conditions necessary to discriminate the classes.

*http://mips.gsf.de/proj/yeast/catalogues/funcat/

	no. rules	no. classes represented	av rule accuracy	average rule coverage (genes)
level 1	159	9	62%	20
level 2	74	12	49%	11
level 3	9	2	25%	18
level 4	37	1	71%	28

Table 4.2: General statistics for rules that appeared more than 5 times. Surprisingly high accuracy at level 4 is due to very few level 4 classes, with one dominating class.

number of rules	class no	class name
17	1/0/0/0	METABOLISM
32	3/0/0/0	CELL GROWTH, CELL DIVISION AND DNA SYNTHESIS
3	4/0/0/0	TRANSCRIPTION
1	5/0/0/0	PROTEIN SYNTHESIS
2	6/0/0/0	PROTEIN DESTINATION
1	7/0/0/0	TRANSPORT FACILITATION
21	9/0/0/0	CELLULAR BIOGENESIS (proteins are not localized to the corresponding organelle)
5	11/0/0/0	CELL RESCUE, DEFENSE, CELL DEATH AND AGEING
77	30/0/0/0	CELLULAR ORGANIZATION (proteins are localized to the corresponding organelle)

Table 4.3: Number of rules that appeared more than 5 times at level 1, broken down by class. Classes not shown had no rules (2/0/0/0, 8/0/0/0, 10/0/0/0, 13/0/0/0 and 90/0/0/0).

---

if the gene is sensitive to calcofluor white
and the gene is sensitive to zymolyase
then its class is "biogenesis of cell wall (cell envelope)"

Mean accuracy: 90.9%
Prior prob of class: 9.5%
Std dev accuracy: 1.8%
Mean no. matching genes: 9.3

---

if the gene is resistant to calcofluor white
then its class is "biogenesis of cell wall (cell envelope)"

Mean accuracy: 43.8%
Prior prob of class: 9.5%
Std dev accuracy: 14.4%
Mean no. matching genes: 6.7

Figure 4.2: Rules regarding sensitivity and resistance to calcofluor white

This was expected, especially since most mutants were only sensitive/resistant to a few media. Some classes were far easier to recognise than others. For example, many good rules predicted class "CELLULAR BIOGENESIS" and its subclass "biogenesis of cell wall (cell envelope)".

The full set of rules can be seen at http://www.aber.ac.uk/compsci/Research/bio/dss/phenotype/ along with the data sets used.

The 4 most frequently appearing rules at level 1 (the most general level in the functional catalogue) are all predictors for the class "CELLULAR BIOGENESIS". These rules suggest that sensitivity to zymolase or papulacandin b, or any reaction (sensitivity or resistance) to calcofluor white is a general property of mutants whose deleted genes belong to the CELLULAR BIOGENESIS class. All correct genes matching these rules in fact also belong to the subclass "biogenesis of cell wall (cell envelope)". The rules are far more accurate than the prior probability of that class would suggest should occur by chance.

Two of these rules regarding sensitivity/resistance to calcofluor white are presented in Figure 4.2. These rules confirm that calcofluor white is useful for detecting cell wall mutations (Ram *et al.*, 1994; Lussier *et al.*, 1997). Calcofluor white is a negatively charged fluorescent dye that does not enter the cell wall. Its main mode of action is believed to be through binding to chitin and prevention

```
if the gene is sensitive to hydroxyurea
then its class is "nuclear organization"

Mean accuracy:  40.2%
Prior prob of class:  21.5%
Std dev accuracy:  6.6%
Mean no.  matching genes:  33.4
```

Figure 4.3: Rule regarding sensitivity to hydroxyurea

of microfibril formation and so weakening the cell wall. The explanation for disruption mutations in the cell wall having increased sensitivity to calcofluor white is believed to be that if the cell wall is weak, then the cell may not be able to withstand further disturbance. The explanation for resistance is less clear, but the disruption mutations may cause the dye to bind less well to the cell wall. Zymolase is also known to interfere with cell wall formation (Lussier *et al.*, 1997). Neither rule predicts the function of any gene of currently unassigned function. This is not surprising given the previous large scale analysis of calcofluor white on mutants.

One rule that does predict a number of genes of unknown function is given in Figure 4.3. This rule predicts the class "nuclear organization" if the gene is sensitive to hydroxyurea. It predicts 27 genes of unassigned function. The rule is not of high accuracy but it is statistically highly significant. Hydroxyurea is known to inhibit DNA replication (Sugimoto *et al.*, 1995), so the rule makes biological sense.

Many genes of unassigned function were predicted by these rulesets. Table 4.4 shows the total number of genes of unassigned function predicted by the learnt rules at levels 1 and 2 in the functional hierarchy. As the rules vary in accuracy, these are plotted as a function of the estimated accuracy of the predictions and the significance (how many standard deviations the estimated accuracy is from the prior probability of the class). As we used a bootstrap process to generate the ruleset, these figures record genes predicted by rules that have appeared more than 5 times during the bootstrap process.

It can be seen that analysis of the phenotype growth data allows the prediction of the functional class of many of the genes of currently unassigned function.

The multi-label version of C4.5 can be compared against the alternative of learning many individual classifiers and combining the results. We took each class in turn and made binary C4.5 classifiers (for example, a classifier that could predict either class "1/0/0/0" or "not 1/0/0/0"). For comparison purposes, the bootstrap

Level 1

estimated accuracy	std. deviations from prior 2	3	4
$\geq 80\%$	83	72	35
$\geq 70\%$	209	150	65
$\geq 50\%$	211	150	65

Level 2

estimated accuracy	std. deviations from prior 2	3	4
$\geq 80\%$	63	63	63
$\geq 70\%$	77	77	77
$\geq 50\%$	133	126	126

Table 4.4: Number of genes of unknown function predicted at levels 1 and 2 in the functional class hierarchy. The number of genes predicted depends on the accuracy and statistical significance demanded.

class	no. rules		av acc		av cov		max acc	
	multi	indiv	multi	indiv	multi	indiv	multi	indiv
1/0/0/0	17	24	42.63	41.07	11.48	7.95	60.00	64.60
2/0/0/0	-	1	-	12.60	-	9.00	-	12.60
3/0/0/0	32	25	52.52	54.40	5.70	6.02	78.40	82.20
4/0/0/0	3	8	30.70	36.37	6.00	5.55	41.00	48.90
5/0/0/0	1	1	21.40	25.70	1.62	1.00	21.40	25.70
6/0/0/0	2	-	22.20	-	3.98	-	32.80	-
7/0/0/0	1	-	12.50	-	7.50	-	12.50	-
8/0/0/0	-	4	-	23.48	-	8.36	-	29.50
9/0/0/0	21	14	66.29	69.03	12.16	13.68	90.30	88.20
11/0/0/0	5	1	33.74	52.70	5.85	6.22	64.00	52.70
30/0/0/0	77	56	74.11	75.46	32.72	48.99	91.10	87.00

Table 4.5: Comparison of the multi-label version of C4.5 against ordinary C4.5 run on each class individually. The bootstrap resampling method was used in each case. "av acc" is the average accuracy of the rules for this class. "av cov" is the average number of ORFs covered by the rules for this class. "max acc" is the maximum accuracy of a rule for this class.

resampling method was used in all cases to produce rulesets. The results for rules that appeared more than 5 times out of the 500 can be seen in Table 4.5. The multi-label version of C4.5 has almost identical results to the results produced by individual classifiers. However, the multi-label version works automatically, in one pass of the data, whereas making individual classifiers requires preprocessing of the data for each class, and learning separate classifiers. This is feasible for the small number of classes at level 1, but becomes time consuming at the lower levels where there are many classes. The multi-label version tends to produce more rules than the binary classifiers, but these extra rules are either variants of the existing rules, or are rules of low accuracy. Whether these extra rules of low accuracy are biologically interesting, or just artifacts of the sparse data and the bootstrap process, remains to be seen.

4.7 Conclusion

In summary our aim of this experiment was to use the phenotype data to discover new biological knowledge about:

- the biological functions of genes whose functions are currently unknown
- the different discriminatory power of the various growth conditions under which the phenotype experiments are carried out

For this we have developed a specific machine learning method which handles the problems provided by this dataset:

- many classes
- multiple class labels per gene
- the need to know accuracies of individual rules rather than the ruleset as a whole

This was an extension of C4.5 coupled with a rule selection and bootstrap sampling procedure to give a clearer picture of the rules.

Biologically important rules were learnt which allow the accurate prediction of functional class for approximately 200 genes. The prediction rules can be easily comprehended and compared with existing biological knowledge. The rules are also useful as they show future experimenters *which media provide the most discrimination between functional classes*. Many types of growth media were shown to be highly informative for identifying the functional class of disruption mutants, while others were of little value. The nature of the C4.5 algorithm is always to choose attributes which split the data in the most informative way. This knowledge can be used in the next round of phenotypic experiments.

Chapter 5

Determining function by expression data

Most of the work in this chapter has been published as Clare & King (2002a).

5.1 Expression data

Another source of data which has recently become widely available and which promises genome-scale answers is expression data. Expression data measure the relative levels of "expression" (production) of RNA in the cell. Since genes are transcribed into proteins via mRNA, this is a way to measure which genes are actively producing proteins. Expression data can be sampled over a period of time, whilst the cell undergoes particular environmental conditions, in order to determine which genes are turned on and off at which times. For example the cell might be subject to a heat shock, and the expression levels monitored before and after, to see which genes are active in dealing with the shock.

Expression data can be collected in a variety of ways. The most common of these is by the use of "microarrays" (Gerhold *et al.*, 1999; Gershon, 2002). Microarrays are tiny chips (for example made of glass), onto which are attached pieces of single-stranded DNA that are complementary to each of the genes under investigation. These pieces of DNA can be attached by photolithography, mechanical spotting, or ink jetting. When the mRNA from the cell is passed over the chip it will bind to the appropriate spots, due to the complementary base pairing. A common approach is to measure expression levels relative to a control sample. The mRNA of the experiment sample is tagged with a red fluorescent dye, and the mRNA of the control is tagged with a green fluorescent dye:

- If a gene is expressed more by the experiment sample than by the control, the spot will appear red.

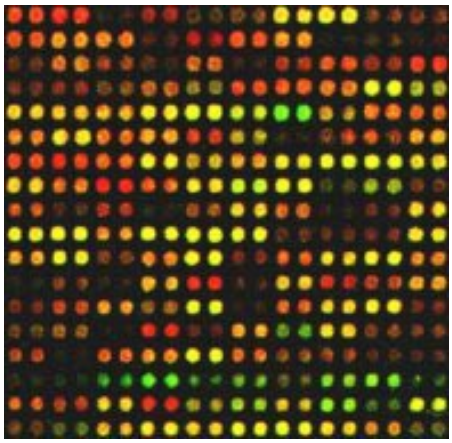


Figure 5.1: A microarray chip. Fluorescent dyes indicate overexpression or underexpression of particular genes.

- If the gene is underexpressed in the experiment it will appear green.
- If the expression of both the control and experiment are equal, mRNA from both will hybridise and it will appear yellow.
- If there is no expression of a particular gene, then this spot will remain black.

Figure 5.1 shows an example of the image produced from a microarray.

There are many reviews of microarray technology and methods for processing expression data in the special supplement to Nature Genetics "The Chipping Forecast" (Various, 1999). Alternatively, introductory information can be found at several websites[1].

Gene expression data is commonly found as time series data. For each gene, the relative expression levels will be reported over a period of time, usually as about a dozen readings for each different environmental condition. Machine learning from such short timeseries is not common, as most learning from time series has been about learning patterns or episodes which can be used to predict future events (for example in stock market data). This type of learning requires much longer time series to provide training data. Straightforward attribute-value learners such as C4.5 can be used, but these treat each time point as independent and unrelated from the others. ILP could be used to describe the relations between the time points, but few ILP systems handle real-valued numerical data well, and most require discretisation of the values which would lose too much of the information.

[1]For example, http://www.gene-chips.com/ and http://www.cs.wustl.edu/~jbuhler/research/array/

Some ILP systems do have the ability to handle real-valued data, for example Aleph (Srinivasan & Camacho, 1999).

The most frequent method used to analyse microarray data is therefore unsupervised learning in the form of clustering. The assumption that genes that share the same expression patterns share a common biological function is the underlying rationale, and this is known as *guilt-by-association* (Walker *et al.*, 1999).

5.2 Decision tree learning

Simple attribute-value learners such as C4.5 can be used on the expression data. On initial runs, we found the results were not as good as the enthusiasm about microarray data promised. A few rules were found, but not many given the amount of data. Because of the problems mentioned previously in Chapter 4 on phenotype data (many classes with not enough data per class) accuracy would be improved by using the bootstrap method of sampling for rules. But collecting together the rules from different rulesets becomes a problem, since this time we have continuous real-valued data. This means that each time we make a ruleset the rules may be slightly different from the previous rules, with slightly different values in the preconditions. For example, the rules in Figure 5.2 are so similar, they really should be counted as the same rule. For this we need to discretise the values of the data.

---

if the gene has a log ratio for cdc28 at 100 minutes of less than -3.14
then its class is ‘‘Protein synthesis’’

---

if the gene has a log ratio for cdc28 at 100 minutes of less than -3.11
then its class is ‘‘Protein synthesis’’

Figure 5.2: Two highly similar rules that should be counted as if they were the same rule.

Discretising into uniform sized bins was the simplest choice, but gave very poor results, as the subtleties of boundaries are lost inside the bins and fewer useful rules were picked up.

Another discretisation algorithm has been proposed by Fayyad and Irani (1993). This algorithm has become very popular and has been has been analysed and compared with other methods in papers such as Kohavi and Sahami (1996). It is

entropy-based and works on the same principles as used by C4.5. This algorithm continuously splits the data, deciding where to split each time by choosing the value that minimises the entropy of the data. The stopping criterion is based on the minimum description length principle. The split points become the discretisation thresholds.

This algorithm appeared ideal as the discretisation boundaries should be very suitable for use with C4.5, and it can be adapted in the same way to cope with multiple class labels. However, in practice, the algorithm failed to make any useful splits of the data, and on further analysis it became clear that splitting the data into subsets actually slightly *increased* the entropy rather than decreased it. This is due to having so many classes and having noisy data.

Experiments on artificial data showed that entropy-based discretisation is highly sensitive to noise. On an artificial data set we constructed (700 examples, 4 classes) the entropy-based method showed poor discretisation with just 5% added noise.

Therefore although decision tree learning was useful and did produce results, it did not seem well suited to the problem: decision tree learning treats each attribute as independent, ignoring the time series relationship between points in this type of data.

5.3 Inductive logic programming

Research in temporal logics seemed promising for use with ILP systems (Baudinet *et al.*, 1993). There have been few attempts to use ILP in the analysis of time series data.

- The use of ILP for time series data was first introduced in 1991 (Feng, 1991). This work used Golem (Muggleton & Feng, 1990) to look at data about the status of a satellite. For this purpose he introduced a new predicate **succeed/2** to relate time points that immediately followed each other. This produced good rules, though their data set was small.

- In 1996, Lorenzo (1996) described an application of ILP using Claudien (De Raedt & Dehaspe, 1997) to a temporal clinical database, defining specific new predicates such as **elapsed-time-max-min/3** and **last-positive-analysis/3** that they thought suitable for capturing features in that domain.

- Badea (2000) described a system for learning rules in the stock market field, learning rules which classified local extrema in the price fluctuations as points at which to buy or sell. Progol was used as the underlying algorithm.

- Also that year Rodríguez *et al.* (2000) described using ILP to classify time series. They did not use an existing ILP algorithm, and instead, defined

their own, specific to this task. They defined extra interval-based predicates such as **always** and **sometimes** and then used a top down method of adding literals to an overly general clause to produce rules. They evaluated their algorithm on several data sets from the UCI archives with good results.

We tried initial experiments of adding simple temporal predicates to the expression data and using Aleph. Using simple predicates such as **successor** gained no more useful rules (perhaps differences across single time steps are not enough to capture the shape of these short time series). Or perhaps the discretisation lost vital information. Adding more complex predicates (**later**, **followed_by**, **peak_at**, **difference_between**, etc) made the search space too large to be tractable. Unfortunately, time constraints of this PhD meant that this work was not taken any further. Further investigation would be desirable into better use of ILP with this data.

5.4 Clustering

Clustering of expression data has been applied in many forms, including hierarchical clustering (top down and bottom up), Bayesian models, simple networks based on mutual information or jackknife correlation, simulated annealing, k-means clustering and self-organising maps (Alon *et al.*, 1999; Eisen *et al.*, 1998; Barash & Friedman, 2001; Butte & Kohane, 2000; Heyer *et al.*, 1999; Lukashin & Fuchs, 2001; Tavazoie *et al.*, 1999; Törönen *et al.*, 1999). Given the wealth of previous work in this area, this seemed a more promising avenue for making use of this data. However, after we applied clustering algorithms to the yeast data, the results did not seem as good as we were led to expect, and we decided to test systematically the validity of the clustering of microarray data.

Most papers on expression data clustering report a selection of good clusters which the authors have *selected by hand* and which correspond to known biology. However, microarray clustering experiments also generally produce clusters which seem to correspond less well with known biology. This type of cluster has received less attention in the literature.

Two main approaches have been employed in testing the reliability of microarray clusters: self-consistency, and consistency with known biology.

- In self-consistency the idea is to predict some left out information in a cluster. For example Yeung *et al.* (2001) used self-consistency to assess the results of their clustering by leaving out one experimental condition (or time point), and using this held out condition to test the predictive ability of the clusters. They were testing whether the cluster could predict the value of the missing experimental condition, or the value at the missing time point.

- The idea behind the use of existing biological knowledge is that if a cluster is consistent with known biological knowledge then it reflects a real feature in the data. This is essentially a systematic version of the informal approach generally taken to evaluate clustering. This idea was used by Tavazoie *et al.* (1999) who clustered the *S. cerevisiae* cdc28 dataset (Cho *et al.*, 1998) with k-means clustering (k=30), and checked the validity of their clustering by mapping the MIPS functional classes onto the clusters to show which classes were found more often than by chance in each cluster. Several clusters were significantly enriched for one or more classes.

In the following experiments we combine the ideas of self-consistency and using known biological knowledge to test systematically the relationship between microarray clusters and known biology. We examine, for a range of clustering algorithms, the quantitative self-consistency of the functional classifications in the clusters.

5.4.1 Microarray Data

To test our approach we used the classic microarray data from Spellman *et al.* (1998), which included 4 different experiments measuring cell-cycle expression levels in the *S. cerevisiae* genome: alpha-factor based synchronisation, cdc15-based synchronisation, elutrition-based synchronisation and the cdc28-based data from Cho *et al.* (1998). To validate this approach and to show that these results are not specific to this dataset of *S. cerevisiae* we also used the data from Khodursky *et al.* (2000) for *E. coli*. This data measured expression levels in response to changes in tryptophan metabolism. Then to show that the general trends are also true of more recent yeast data sets we used the data set from Gasch *et al.* (2000) which measured the expression levels of yeast cells when subject to a wide variety of environmental changes.

5.4.2 Classification Schemes

For *S. cerevisiae* we selected the most commonly used functional classification schemes: the Munich Information Center for Protein Sequences (MIPS) scheme[2], and the GeneOntology (GO) consortium scheme[3]. For *E. coli* we used GenProtEC's MultiFun classification scheme[4]. Section 2.6 described these classification schemes in more detail.

[2]http://mips.gsf/proj/yeast/catalogues/funcat
[3]http://www.geneontology.org
[4]http://genprotec.mbl.edu/start

5.4.3 Clustering Methods

We chose three clustering methods to compare: agglomerative hierarchical clustering (Eisen *et al.*, 1998), k-means clustering (Duda *et al.*, 2000), and a modified version of the method of Heyer *et al.* (Heyer *et al.*, 1999) "QT_CLUST".

Hierarchical and k-means clustering are the two methods currently available on the EBI's Expression Profiler[5] web server. The Expression Profiler is an up to date set of tools available over the web for clustering, analysis, and visualisation of microarray data.

- Hierarchical clustering is an agglomerative method, joining the closest two clusters each time, then recalculating the inter-cluster distances, and joining the next closest two clusters together. We chose the average linkage of Pearson correlation as the measure of distance between clusters, and a cut-off value of 0.3.

- K-means clustering is a standard clustering technique, well known in the fields of statistics and machine learning. Correlation was used as distance measure. We used k=100.

- The QT_CLUST algorithm is described in (Heyer *et al.*, 1999). We implemented a modified version of this algorithm. The data was normalised so that the data for each ORF had mean 0 and variance 1. Pearson correlation was used as a similarity measure, and each ORF was used to seed a cluster. Further ORFs were added to the cluster if their similarity with all ORFs in the cluster was greater than a fixed threshold (the cluster diameter). We used a cluster diameter (minimum similarity) of 0.7. This was taken to be our final clustering. We did not implement Heyer *et al.*'s method of removal of outliers by computing jackknife correlations. Also, we did not, as Heyer *et al.* do, set aside the largest cluster and recluster, since we did not demand a unique solution, and in fact we wanted a clustering in which each ORF could be in more than one cluster. Allowing each ORF to be in more than one cluster is similar to the situation when using the functional hierarchies as ORFs can belong to more than one functional class. There can be as many clusters as there are ORFs - there is no fixed number of clusters which has to be decided beforehand or as part of the training process. We henceforth refer to this method as QT_CLUST_MOD.

These three methods were chosen not to show that any one was better than any other (this would be difficult to prove, given the variety of parameters that can be tuned in each algorithm), but rather, to give a representative sample of

[5]http://ep.ebi.ac.uk/

commonly used clustering algorithms for microarray data - to show that what we observe is approximately true for any reasonable clustering.

All clustering parameters (hierarchical cut-off value, k means, cluster diameter) were chosen to be reasonable after experimentation with various values. Although parameters could possibly have been further refined, this was not the aim of this work.

5.4.4 Predictive Power

We required a quantitative measure of how coherent the ORF clusters formed from microarray expression data are with the known functions of the ORFs.

We form this measure as follows: in the k-means and hierarchical clusterings, each ORF appears in only one cluster, so we can take each ORF in turn and test whether or not the cluster without this ORF can predict its class; for the QT_CLUST_MOD scheme, the clusters are seeded by each ORF in turn, and we test for each seed in turn whether or not the rest of the cluster predicts the class of the seed. That is, we test whether or not the majority class of the cluster is one of the classes of the heldout ORF. The majority class is the class most frequently found among the ORFs in the cluster. We call this measure the 'Predictive Power' of the clustering. Tests of predictive power were only carried out on clusters which had more than 5 ORFs, and only on ORFs which were not classified as unknown.

For example, if a cluster contained ORFs belonging to the following classes:

```
ORF  Class
orf1  A
orf2  B
orf3  A
orf4  A
orf5  A
orf6  A
orf7  A
orf8  A
orf9  B
orf10 A
```

then the majority class of the cluster without orf1 would be "A", which is a correct prediction of the actual class of orf1. However orf2 and orf9 would be incorrectly predicted by this cluster. Class "A" is correctly predicted in 8/10 cases. (Class "B" is never predicted by this cluster, since it is never the majority class).

To test the statistical significance of this measure we used a Monte Carlo type approach. ORFs were chosen at random, without replacement, and random

clusters formed using the same cluster size distribution as was observed in the real clusterings. The resulting random clusters were then analysed in the same manner as the real clusters. A thousand random clusterings were made each time, and both the mean results reported, and how many times the random cluster accuracy for a functional class was equal or exceeded that of the actual clusters.

5.4.5 Results

The quality of the clusters produced by the different programs was, we believe, consistent with those shown in previous results. Initial inspection of the clusters showed some obviously good groupings, and other clusters were produced which generally seemed to have something in common, but the signal was less strong. *However, most clusters on inspection did not appear to share anything in common at all.*

An example of a strong cluster in shown in table 5.1. The probability of this cluster occurring by chance is estimated to be less than $2 * 10^{-17}$ (calculated using the hypergeometric distribution). However, note the mannose related sub-cluster in this cluster. The most likely explanation for the sub-cluster is: as the yeast data was formed to study cell division, histones and mannose are both required in the same time specific pattern during division. They therefore either share the same transcription control mechanism, or have mechanisms that are commonly controlled. This hypothetical common transcriptional control in cell division is not reflected in the current annotation.

Clusters which appeared to be unrelated were common. There were also clusters which did seem to contain related ORFs, but less obviously than the histone cluster mentioned above. Table 5.2 shows an example of a cluster which does show a DNA processing theme, but this theme is not reflected in the variety of classifications of the ORFs.

To quantitatively test the relationship between clusters and annotations we evaluated all the clusters formed using our measure of *predictive power* (see Table 5.3). For *S. cerevisiae* we calculated the predictive power of each clustering method (k-means, hierarchical, and QT_CLUST_MOD) for each functional class in levels 1 and 2 of MIPS and GO. For *E. coli* we calculated this for each functional class in level 1 of the GenProtEC hierarchy. This produced a large number of annotated clusterings, and the complete set of these can be found at http://www.aber.ac.uk/compsci/Research/bio/dss/gba/. The same broad conclusions regarding the relationship between clusters and annotation were true for both species and using all clustering methods; we have therefore chosen to present the *S. cerevisiae* tables only for hierarchical clustering and for only the first levels of the GO and MIPS annotation hierarchies, and one table for *E. coli*. The predictive power of the different clustering methods can be seen in Tables 5.4, 5.5, and 5.6. The results

are broken down according to the majority classes of the clusters. Absence of data for a class indicates that this class was not the majority class of any cluster.

All the clustering methods produced statistically significant clusters with all three functional annotation schemes. This confirms that clusters produced from microarray data reflect some known biology. However, the predictive power of even the best clusters of the clearest functional classes is low (mostly $< 50\%$). This means that if you predict function based on guilt-by-association your predictions will contradict existing annotations a large percentage of the time.

One of the clearest messages from the data is the large difference in predictive power across the different microarray experiments, clustering methods, and annotation schemes. There is no clear best approach and quite different significance results are obtained using different combinations.

Perhaps the most interesting differences are those between different microarray experiments: alpha, cdc15, cdc28, and elu. It is to be expected that different microarray experiments will highlight different features of cell organisation. However it is unclear how biologically significant these differences are. Using the GO annotation scheme:

- alpha is best for predicting classes: enzyme; nucleic acid; chaperone; motor; and cell-adhesion.
- cdc15 is best for predicting classes: transporter; and ligand binding or carrier.
- cdc28 is best for predicting class: structural protein.

Using the MIPS annotation scheme:

- alpha is best for predicting class: protein destination.
- cdc15 is best for predicting classes: metabolism; cell growth, cell division and DNA synthesis; and transport facilitation.
- cdc28 is best for predicting classes: cellular organisation; transcription; and protein synthesis.
- elu is best for class cellular transport and transport mechanism.

A particularly dramatic difference is that for the GO class "ligand binding or carrier" using hierarchical clustering, where the cdc15 and elu experiments produced highly significant clusters whereas the alpha and cdc28 experiments produced clusters with negative correlation.

The classes highlighted also differed significantly between clustering methods. Considering first the GO annotation scheme: the clustering method k-means is best for predicting enzyme class; QT_CLUST_MOD is best for the classes "nucleic

acid binding", "chaperone", and "cell-adhesion"; and hierarchical clustering is best for "structural protein", "transporter", and "ligand binding or carrier chaperone". Considering the MIPS annotation scheme: the clustering method k-means is best for predicting classes "cell rescue, defence, cell death and ageing", and "protein synthesis"; QT_CLUST_MOD is best for the class "transcription"; and hierarchical clustering is best for "cell organisation", "metabolism", "cell growth", "cell division" and "DNA synthesis".

The data also revealed some unexpected apparent negative correlations between clusters and classes. For example using the MIPS annotation scheme and hierarchical clustering to cluster the cdc28 data, the random clustering produced a higher predictive power $> 95\%$ of the time for classes "cellular transport" and "transport mechanism". Transport proteins seem particularly poorly predicted both in both *S. cerevisiae* and *E. coli*. A possible explanation for this is that their transcription control is synchronised with the specific pathways they are involved with rather than as a group.

Do the two annotation schemes of MIPS and GO agree on cluster consistency? Sometimes, with strong clusters, such as within the ribosomal clusters (see Figure 5.3). But on the whole, the correlation between the scores given by the two annotation schemes is approximately 0. This is partly due to the fact that GO currently has many fewer annotations than MIPS, so there are several clusters which show a trend under MIPS annotation that cannot be seen under GO, because too many ORFs have no annotation.

Are the annotation schemes improving over time with respect to these clusters? Tables 5.7 and 5.8 show the accuracies of the clusters found by hierarchical clustering under MIPS annotations. Table 5.7 uses the MIPS annotations from 21st December 2000, whereas Table 5.8 uses the MIPS annotations from 24th April 2002, 16 months later. The accuracies are almost identical.

Does simple preprocessing help? Table 5.9 shows a comparison of accuracy when normalisation or removal of ORFs with low standard deviation was used. There is no consistent trend of improvement or degradation and very little difference between the results.

It has been commented that perhaps other distance measures or a different choice of linkage could be more appropriate for the use of hierarchical clustering on expression data. We show that use of Euclidean distance and complete linkage does little to change the accuracy and in fact seems worse than correlation and average linkage for the alpha dataset. This can be seen in Table 5.10 by comparison to Table 5.9.

5.4.6 Discussion

Given a clustering produced from microarray data and a protein functional classification scheme there are four possibilities:

- the annotations confirm the cluster.
- the annotations and the cluster disagree because the microarray data involves biological knowledge not explicitly represented in the annotations.
- the annotations and the cluster disagree because the microarray data involves new biological knowledge.
- the annotations and the cluster disagree because the cluster is noise.

We have quantified how often the first case occurs and illustrated the limitations of existing annotations in explaining microarray data (Kell & King, 2000). These clusters where annotation and microarray data agree are the first choice of clusters to examine to gain knowledge of control of transcription. In favour of the second explanation is the fact the functional classification schemes are still "under construction" and do not reflect all that is known about biology. There are also many possible improvements in clustering algorithms which could improve the consistency. It is to be expected that the third case will predominate. Microarrays are a fascinating technology and an industry has been based on them. It is almost inconceivable that microarrays will not reveal large amounts of new and fascinating biological knowledge.

A major challenge in microarray analysis is therefore to discriminate between the old and new biology in the data and the noise. To achieve this we require

- Better models of instrument noise in microarrays (Newton *et al.*, 2001).
- Data analysis methods explicitly designed to exploit time-series data (with most existing clustering methods you could permute the time points and get the same results).
- Ways of combining information from different experiments to provide repeatability (known standards of data will help this, such as MIAME (MGED Working Group, 2001)).
- Deeper data analysis methods designed to elucidate the biological processes behind the clusters, such as genetic networks.

5.5 Summary

In this chapter learning from expression data was investigated. C4.5 was tried but few good rules were discovered, and the data was too noisy for good discretisation. ILP (Aleph) was investigated as a way to represent the relationships between timepoints, but the search space for interesting clauses was too large to be tractable,

and time limitations prevented further analysis. Clustering algorithms were investigated as these are the most common way of analysing expression data in the literature. The clusters produced were found to be consistent with class groupings only in certain cases, and the majority of clusters did not agree with the functional classes. Microarray chips are a new technology and expression data is currently noisy and error prone. Better standards of data are needed in future experiments, and more work is needed to determine good machine learning techniques for this data.

ORF	description	MIPS classes
ybr008c	fluconazole resistance protein	11/7/0/0 7/28/0/0
ypl127c	histone H1 protein	30/10/0/0 30/13/0/0
ynl031c	histone H3	30/10/0/0 30/13/0/0 4/5/1/4
ynl030w	histone H4	30/10/0/0 30/13/0/0 4/5/1/4
ylr455w	weak similarity to human G/T mismatch binding protein	99/0/0/0
ygl065c	mannosyltransferase	1/5/1/0 6/7/0/0
yer003c	mannose-6-phosphate isomerase	1/5/1/0 30/3/0/0
ydr225w	histone H2A	30/10/0/0 30/13/0/0 4/5/1/4
ydr224c	histone H2B	30/10/0/0 30/13/0/0 4/5/1/4
ydl055c	mannose-1-phosphate guanyltransferase	1/5/1/0 9/1/0/0
ybr010w	histone H3	30/10/0/0 30/13/0/0 4/5/1/4
ybr009c	histone H4	30/10/0/0 30/13/0/0 4/5/1/4
ybl003c	histone H2A.2	30/10/0/0 30/13/0/0 4/5/1/4
ybl002w	histone H2B.2	30/10/0/0 30/13/0/0 4/5/1/4

Table 5.1: A yeast histone cluster (cdc15 data, QT_CLUST_MOD clustering algorithm, MIPS annotations, cluster id: 203)

ORF	description	MIPS classes
ycl064c	L-serine/L-threonine deaminase	1/1/10/0
yol090w	DNA mismatch repair protein	3/19/0/0 30/10/0/0
yol017w	similarity to YFR013w	99/0/0/0
ynl273w	topoisomerase I interacting factor 1	99/0/0/0
ynl262w	DNA-directed DNA polymerase epsilon, catalytic subunit A	11/4/0/0 3/16/0/0 3/22/1/0 30/10/0/0
ynl082w	DNA mismatch repair protein	3/19/0/0 30/10/0/0
ynl072w	RNase H(35), a 35 kDa ribonuclease H	1/3/16/0
ylr049c	hypothetical protein	99/0/0/0
yjl074c	required for structural maintenance of chromosomes	3/22/0/0 9/13/0/0
yhr153c	sporulation protein	3/10/0/0 3/13/0/0
yhr110w	p24 protein involved in membrane trafficking	6/4/0/0 8/99/0/0
ygr041w	budding protein	3/4/0/0
ydl227c	homothallic switching endonuclease	3/7/0/0 30/10/0/0
ydl164c	DNA ligase	11/4/0/0 3/16/0/0 3/19/0/0 30/10/0/0
ydl156w	weak similarity to Pas7p	99/0/0/0
ybr071w	hypothetical protein	99/0/0/0
yar007c	DNA replication factor A, 69 KD subunit	3/16/0/0 3/19/0/0 3/7/0/0 30/10/0/0

Table 5.2: A yeast DNA processing cluster. Note the variety of MIPS classes represented here. (cdc28 data, QT_CLUST_MOD clustering algorithm, MIPS annotations, cluster id: 599)

```
MIPS

ydr417c  questionable ORF                              99/0/0/0
ylr325c  60S large subunit ribosomal protein           30/3/0/0 5/1/0/0
yjl177w  60s large subunit ribosomal protein L17.e     30/3/0/0 5/1/0/0
yml063w  ribosomal protein S3a.e                       30/3/0/0 5/1/0/0
ydr447c  ribosomal protein S17.e.B                     30/3/0/0 5/1/0/0
ykl056c  strong similarity to human IgE-dependent histamine-releasing
                                          factor  30/3/0/0 98/0/0/0
ylr061w  ribosomal protein                             5/1/0/0
ykr094c  ubiquitin                              30/3/0/0 5/1/0/0 6/13/1/0
yol039w  acidic ribosomal protein P2.beta              30/3/0/0 5/1/0/0
ydr418w  60S large subunit ribosomal protein L12.e     30/3/0/0 5/1/0/0
ykl006w  ribosomal protein                             30/3/0/0 5/1/0/0
yol040c  40S small subunit ribosomal protein           30/3/0/0 5/1/0/0
yor167c  40S small subunit ribosomal protein S28.e.c15 30/3/0/0 5/1/0/0
ylr367w  ribosomal protein S15a.e.c12                  30/3/0/0 5/1/0/0
ypr102c  ribosomal protein L11.e                       30/3/0/0 5/1/0/0
ypr118w  similarity to M.jannaschii translation initiation
                                       factor, eIF-2B  99/0/0/0
--------
GO

ydr417c  molecular_function unknown       GO_0005554
ylr325c  structural protein of ribosome   GO_0005198 :  GO_0003735
yjl177w  structural protein of ribosome   GO_0005198 :  GO_0003735
yml063w  structural protein of ribosome   GO_0005198 :  GO_0003735
ydr447c  structural protein of ribosome   GO_0005198 :  GO_0003735
ykl056c  molecular_function unknown       GO_0005554
ylr061w  structural protein of ribosome   GO_0005198 :  GO_0003735
ykr094c  structural protein of ribosome   GO_0005198 :  GO_0003735
yol039w  structural protein of ribosome   GO_0005198 :  GO_0003735
ydr418w  structural protein of ribosome   GO_0005198 :  GO_0003735
ykl006w  structural protein of ribosome*  GO_0003676 GO_0005198 :
                                               GO_0003723 GO_0003735
yol040c  structural protein of ribosome   GO_0005198 :  GO_0003735
yor167c  structural protein of ribosome   GO_0005198 :  GO_0003735
ylr367w  structural protein of ribosome   GO_0005198 :  GO_0003735
ypr102c  structural protein of ribosome   GO_0005198 :  GO_0003735
ypr118w  molecular_function unknown       GO_0005554
```

Figure 5.3: Ribosomal cluster as agreed by MIPS and GO. Semicolons separate the levels of GO classes. They agree on all except ykr094c. (This example is cluster ID: 390, alpha data, hierarchical clustering)

	random	alpha	cdc15	cdc28	elu	*E. coli*
MIPS - hier	56.257	61.062 (0)	62.821 (0)	61.341 (0)	60.270 (0)	-
MIPS - k	59.304	58.795 (989)	59.053 (908)	59.677 (4)	59.583 (17)	-
MIPS - QT	58.256	59.714 (16)	61.697 (0)	62.136 (0)	59.631 (21)	-
GO - hier	52.265	62.526 (0)	60.799 (0)	61.087 (0)	59.344 (0)	-
GO - k	59.301	61.649 (0)	59.990 (1)	62.067 (0)	60.638 (0)	-
GO - QT	55.397	60.799 (0)	60.236 (0)	61.288 (0)	60.146 (0)	-
E. coli - hier	52.906	-	-	-	-	57.785 (0)
E. coli - k	53.104	-	-	-	-	56.103 (0)
E. coli - QT	52.751	-	-	-	-	55.291 (0)

Table 5.3: A summary of the average predictive power for each type of clustering for each experiment. Figures are percentage correct predictions. "random" shows mean over 1000 random clusterings. Figures in brackets show how many times out of 1000 the random clustering produced equal or greater than this percentage.

	random	alpha	cdc15	cdc28	elu
enzyme	59.487	**64.578 (0)**	61.753 (64)	61.771 (61)	60.511 (238)
nucleic acid binding	16.355	26.531 (13)	15.584 (574)	25.439 (22)	22.430 (78)
structural protein	12.767	**50.725 (0)**	**47.423 (0)**	**57.792 (0)**	20.290 (52)
transporter	8.585	13.725 (154)	**32.432 (0)**	10.417 (330)	5.357 (735)
ligand binding or carrier	6.825	4.167 (669)	**30.769 (0)**	3.571 (706)	**21.429 (4)**
chaperone	3.170	13.636 (55)	-	5.263 (278)	-
signal transducer	2.938	14.815 (34)	15.000 (34)	3.703 (303)	11.765 (69)
motor	1.069	-	-	-	11.111 (41)

Table 5.4: **Yeast hierarchical clustering (cut-off=0.3) class by class breakdown at level 1 GO**. First column shows average over 1000 random clusterings. alpha, cdc15, cdc28 and elu are the 4 cell-cycle synchronisation methods. Figures show percentage correct predictions. The figure in brackets is how many times out of 1000 the random clustering produced equal or greater than this percentage. If less than 5, this value is highlighted.

	random	alpha	cdc15	cdc28	elu
cellular organization	59.344	**61.988 (4)**	**63.258 (0)**	**64.442 (0)**	**61.146 (42)**
metabolism	28.827	**39.721 (1)**	**40.136 (0)**	33.951 (82)	37.061 (6)
cell growth, cell division and DNA synthesis	22.429	**43.421 (0)**	**46.961 (0)**	**35.816 (1)**	22.963 (478)
transcription	20.879	23.333 (304)	30.496 (26)	**33.333 (4)**	18.750 (681)
protein destination	14.988	17.021 (350)	18.367 (266)	14.865 (495)	11.842 (710)
cellular transport and transport mechanisms	12.500	12.500 (491)	-	2.273 (957)	21.951 (70)
cell rescue, defense, cell death and ageing	9.495	18.750 (60)	18.605 (60)	21.739 (35)	14.706 (163)
transport facilitation	8.024	14.815 (135)	**28.889 (2)**	2.703 (792)	-
protein synthesis	8.760	**68.000 (0)**	15.385 (175)	**56.716 (0)**	-
energy	5.891	13.636 (140)	-	7.895 (349)	-
cellular biogenesis	5.241	11.765 (160)	-	6.250 (367)	-
ionic homeostasis	2.805	-	-	-	14.286 (73)

Table 5.5: **Yeast hierarchical clustering (cut-off 0.3) class by class breakdown at level 1 MIPS**. First column shows average over 1000 random clusterings. alpha, cdc15, cdc28 and elu are the 4 cell-cycle synchronisation methods. Figures show percentage correct predictions. The figure in brackets is how many times out of 1000 the random clustering produced equal or greater than this percentage. If less than 5, this value is highlighted.

	hier	k	QT
metabolism	**56.579 (0)**	**55.581 (0)**	**58.115 (0)**
location of gene products	**57.107 (0)**	**55.037 (4)**	**56.513 (0)**
cell structure	**59.794 (0)**	34.884 (294)	30.755 (509)
information transfer	35.417 (116)	26.315 (341)	16.667 (699)
regulation	36.364 (68)	-	7.692 (623)
transport	38.095 (69)	-	13.333 (748)
cell processes	57.692 (11)	58.140 (14)	63.636 (8)
extrachromosomal	**64.286 (0)**	39.189 (61)	**42.105 (3)**

Table 5.6: *E. coli* **class by class breakdown at level 1**. QT = QT_CLUST_MOD (0.7), k = k-means clustering (100), hier = hierarchical clustering (0.3). Figures show percentage correct predictions. The figure in brackets is how many times out of 1000 the random clustering produced equal or greater than this percentage. If less than 5, this value is highlighted.

	hier
energy	88.462
cellular organization	65.163
protein destination	62.500
metabolism	56.738
cell growth, cell division and DNA synthesis	48.649
transcription	46.970
transport facilitation	42.105
cellular transport and transport mechanisms	37.500
cellular biogenesis	20.000
cell rescue, defense, cell death and ageing	16.667

Table 5.7: **Gasch data set, MIPS classification as of 21/12/00**. Class by class breakdown at level 1. Hierarchical clustering with cut-off 0.3. Figures show percentage correct predictions.

	hier
energy	79.310
protein fate (folding, modification, destination)	66.667
subcellular localisation	64.158
metabolism	50.794
cell cycle and DNA processing	48.148
transcription	47.692
transport facilitation	42.105
cellular transport and transport mechanisms	35.294
control of cellular organization	25.000
cell rescue, defense and virulence	16.667

Table 5.8: **Gasch data set, MIPS classification as of 24/4/02**. Class by class breakdown at level 1. Hierarchical clustering with cut-off 0.3. Figures show percentage correct predictions.

class	plain	norm	rem low	norm and rem low
energy	21.739	15.789	18.750	18.750
protein fate (folding, modification, destination)	16.102	16.667	18.519	9.756
subcellular localisation	61.079	61.460	62.097	62.267
metabolism	39.432	38.387	37.500	29.208
cell cycle and DNA processing	37.013	36.111	41.026	43.564
transcription	20.800	29.134	16.346	18.447
transport facilitation	11.765	19.355	-	-
cellular transport and transport mechanisms	10.204	10.870	-	-
control of cellular organization	8.696	10.000	15.385	20.000
cell rescue, defense and virulence	21.622	21.622	33.333	22.222
cell fate	16.129	17.544	25.000	14.634
protein synthesis	61.765	67.741	43.478	52.174
regulation of/interaction with cellular environment	-	15.385	14.286	-

Table 5.9: **The effects of preprocessing on accuracy**. Preprocessing of the data is compared. "plain" is a baseline, no preprocessing. "norm" is normalisation where mean and standard deviation are normalised to 0 and 1 respectively for each ORF. "rem low" is removal of ORFs with low standard deviation (in the bottom 25% of the data set). "rem low and norm" is removal of ORFs with low standard deviation followed by normalisation of the remaining ORFs. The dataset was alpha data, MIPS classification as of 24/4/02. Clustering was hierarchical clustering, average linkage of correlation, cut-off=0.3.

	complete linkage, euclidean distance
energy	18.182
protein fate (folding, modification, destination)	24.107
subcellular localisation	60.623
metabolism	34.819
cell cycle and DNA processing	19.388
transcription	35.545
transport facilitation	-
cellular transport and transport mechanisms	10.169
control of cellular organization	7.143
cell rescue, defense and virulence	5.405
cell fate	7.407
protein synthesis	33.766
regulation of/interaction with cellular environment	6.250

Table 5.10: **Use of complete linkage and Euclidean distance for hierarchical clustering**. Compare these values with the "plain column" in Table 5.9. Dataset was alpha data, MIPS classification as of 24/4/02. Clustering was hierarchical clustering, complete linkage of Euclidean distance, cut-off=1.0. (0.3 gave clusters containing a maximum of 2 ORFs only, so was too tight).

Chapter 6

Distributed First Order Association Rule Mining (PolyFARM)

6.1 Motivation

Genomes of ever increasing size are being sequenced. In the year 2000, the sequence of *A. thaliana* was published with its 25,000 genes (Arabidopsis genome initiative, 2000), and a first draft of the human genome was published last year with estimates of between 30,000 and 40,000 genes (International human genome sequencing consortium, 2001; Venter *et al.*, 2001). Two draft sequences of rice genomes were published on 5th April 2002 (Yu *et al.*, 2002; Goff *et al.*, 2002) with estimates of 32,000 to 55,000 genes. If we wish to continue using the relational data mining algorithm WARMR as our preprocessing step, it will need the ability to scale up to such problems. The three main approaches stated by Provost and Kolluri (1999) for scaling up an inductive algorithm are:

- use a relational representation
- design a fast algorithm
- partition the data

WARMR already uses a relational representation. Based on APRIORI, it is already fast, though could perhaps be tuned to fit our particular problem. But to really scale up this algorithm, we must consider the "partition the data" approach and develop a solution that makes use of our parallel processing power.

Recent work on parallel and distributed association rule mining was reviewed in Section 1.5.5. None of this work has been extended to first order association rule mining to the best of our knowledge. Although almost all ILP algorithms

learn rules from first order data, WARMR is currently the only general first order **association rule** mining algorithm available[1].

6.2 WARMR

The basic algorithm of WARMR is a levelwise algorithm, similar to that of AIS and APRIORI (described in Sections 1.5.2 and 1.5.3). The database to be mined is expressed in Datalog. The patterns to be discovered are first order associations or "queries". A query is a conjunction of literals (existentially quantified, but written without the quantifier where it is clear from the context that queries are meant). Examples of queries are:

A pizza that Bill buys and Sam likes:

$$pizza(X) \wedge buys(bill, X) \wedge likes(sam, X)$$

An ORF that is homologous to a protein with the keyword "transmembrane":

$$orf(X) \wedge homologous(X, Y) \wedge keyword(Y, transmembrane)$$

Queries are constructed in a levelwise manner: at each level, new candidate queries are generated by specialisation of queries from the previous level under θ-subsumption. This specialisation is achieved by extension of each of the previous queries by each of the literals in the language allowed by the language bias. Candidate queries are counted against the database and pruned away if their support does not meet the minimum support threshold (θ-subsumption is monotonic with respect to frequency). The surviving candidates become the frequent query set for that level and are used to generate the next level. The algorithm can be used to generate all possible frequent queries, or to generate queries up to a certain length (i.e. level).

WARMR provides a language bias of modes, types and constraints for the user to restrict the search and specify dependencies between literals. The user also specifies a "key" atom. This is one which partitions the database into entities (for example, supermarket baskets, genes, employees or transactions). Frequency is counted with respect to these entities, and they are the focus for the data mining.

Initially we wanted to use WARMR to process the relational yeast data sets. Unfortunately, this was not possible due to the amount of memory required by the system for this data being more than was available under Sicstus Prolog. The

[1]Two other first order mining algorithms exist: RAP (Blat'ák *et al.*, 2002) which is a new system for mining maximal frequent patterns, and MineSeqLog (Lee & De Raedt, 2002), a new algorithm for finding sequential queries.

WARMR team were working on developing a new Prolog compiler (ilProlog), but it was still under development and again did not work on our data.

6.3 PolyFARM

6.3.1 Requirements

What we required for the yeast genome data is a system which counts queries (associations) in relational data, progressing in a levelwise fashion, and making use of the parallel capabilities of our Beowulf cluster (distributed memory, approx 60 nodes, between 256Mb and 1Gb memory per node). We will use Datalog[2] (Ullman, 1988) to represent the database. When the database is represented as a flat file of Datalog facts in plain uncompressed text, each gene has on average 150Kb of data associated with it (not including background knowledge). This is in total approximately 1Gb for the whole yeast genome when represented in this way. Scaling is a desirable feature of any such algorithm. Our software should scale up to larger genomes. It should be robust to changes in the Beowulf configuration. If a node goes down whilst processing we need the ability to recover gracefully and continue. The software should able to make use of additional processors if they are added to the Beowulf cluster in the future, and indeed should not rely on any particular number of processors being available.

The two main options for parallelisation considered by most of the algorithms described in section 1.5.5 are partitioning the query candidates and partitioning the database.

Partitioning the candidate queries: In this case, it is difficult to find a partition of the candidate queries which optimally uses all available nodes of the Beowulf cluster without duplication of work. Many candidates share substantial numbers of literals, and it makes sense to count these common literals only once, rather than repeatedly. Keeping candidates together which share literals makes it difficult to produce a fair split for the Beowulf nodes.

Partitioning the database: The database is more amenable to partitioning, since we have more than 6000 genes, each with their own separate data. Division of the database can take advantage of many Beowulf nodes. Data can be partitioned into pieces which are small enough to entirely fit in memory of a node, and these partitions can be farmed out amongst the nodes,

[2]Datalog is the language of function free and negation free Horn clauses (Prolog without functions). As a database query language it has been extensively studied. Datalog and SQL are incomparable in terms of expressiveness. Recursive queries are not possible in SQL, and Datalog needs the addition of negation to be more powerful than SQL.

with nodes receiving extra partitions of work when they finish. Partitioning the database means that we can use the levelwise algorithm, so to produce associations of length d this requires just d passes through the database. In this application we expect the size of the database to be more of an issue than the size of the candidates.

To answer these requirements we designed PolyFARM (Poly-machine First-order Association Rule Miner). We chose to partition the database and to count each partition independently.

6.3.2 Farmer, Worker and Merger

There are three main parts to the PolyFARM system:

Farmer Reporting of results so far, and candidate query generation for the next level

Worker Candidate frequency counting on a subset of the database

Merger Collation and compaction of Worker results to save filespace

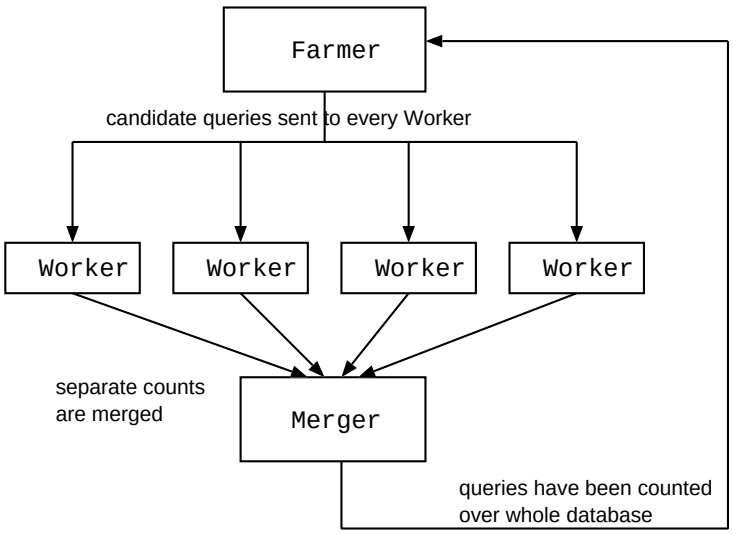


Figure 6.1: Farmer, Worker and Merger

The interactions between the parts are shown in Figure 6.1. The candidate queries are generated just once centrally by Farmer, using the language bias and the frequent queries from the previous level. The database is partitioned, and each Worker reads in the candidates, its own database partition and the common

```
farmer = read_in {settings, background and current queries}
         prune queries
         print results
         specialise queries
         write_out {instructions for workers, new queries}

worker = read_in {database chunk, settings, candidate queries}
         count queries
         write_out {counted queries}

merger [fileN .. fileM] =
         mergeCounts fileN (merger [fileN+1 .. fileM])

  where mergeCounts queries1 queries2 =
              sum the counts from queries1 and queries2
```

Figure 6.2: Processes for Farmer,Worker and Merger

background knowledge. Candidates are evaluated (counted) against the database partition, and the results are saved to file, as the Beowulf has no shared memory, and we do not rely on any PVM-like architectures. When all Workers have completed, the Farmer uses the results produced by the Workers. It prunes away the infrequent queries, and displays the results so far. Then the Farmer generates the next level of candidates, and the cycle begins again. A single Worker represents counting of a single partition of a database. On the Beowulf cluster, each node will be given a Worker program to run. When the node has completed, and the results have been saved to a file, the node can run another Worker program. In this way, even if there are more partitions of the database than nodes in the Beowulf cluster, all partitions can be counted within the memory available.

The one problem with this system, is that in generating a file of counts from each Worker's database partition, so many files can be generated that filespace could become an issue. So we introduce a third step - Merger. Merger collates together Worker files into one single file, saving space. Merger can be run at any time, when filespace needs compacting. Finally, Farmer will simply read in the results from Merger, rather than collating Workers' results itself. The main processes described here for the Farmer, Worker and Merger algorithms are shown in Figure 6.2.

This solution addresses 2 aspects of scaling of the size of the database:

- Memory: Partitioning data for the Workers means that no Worker need

handle more data than can fit in its main memory, no matter how large the database becomes.

- Filespace: Merger means that the buildup of intermediate results is not a filespace issue.

Partitioning the database will not address the problem of growth of candidate space, but will address the problem of searching over a large database. In our application we will be searching for relatively short queries, and so we do not anticipate the size of the candidate query space to be a problem.

6.3.3 Language bias

Many machine learning algorithms allow the user to specify a language bias. This is simply the set of factors which influence hypothesis selection (Utgoff, 1986). Language bias is used to restrict and direct the search. Weber (1998) and Dehaspe (1998) describe more information about declarative language biases for datamining in ILP. Weber (1998) describes a language bias for a first order algorithm based on APRIORI. Dehaspe (1998) describes and compares two different language biases for data mining: DLAB and WRMODE. We will follow the lead of WARMR and allow the user a declarative language bias which permits the specification of modes, types and constraints.

- Modes are often used in ILP. Arguments of an atom can be specified as $+$ (must be bound to a previously introduced variable), $-$ (introduces a new variable), or as introducing a constant. In PolyFARM, constants can be specified in two ways, which are simply syntactic variants for ease of comprehension: *constlist*[...], where the list of available constants is given immediately in the mode declaration (and is a short list), and *constpred predname*, where the constants are to be generated by an arity 1 predicate in the background knowledge file (this case would be chosen when the list of constants is large).

- Types are used to restrict matching between arguments of literals. An argument which has a mode of $+$, must be bound to a previous variable, but this previous variable must be of the same type. For example, if we consider the types:

$$buys(Person, Food).$$
$$student(Person).$$

  and a query $buys(X, Y)$, and a new literal to be added $student(Z)$, we know that Z cannot be bound to Y, since these would be conflicting types. Types restrict the search space to semantically correct solutions.

- Constraints are used when further restrictions are required. Currently constraints can be used to restrict the number of times a predicate can be used in any one query, or to state that one predicate may not be added if another predicate is already in the query. The latter form of constraint can be used to ensure that duplicate queries are not produced through different orders of construction. For example queries $buys(pizza, X) \wedge student(X)$ and $student(X) \wedge buys(pizza, X)$ are equivalent.

6.3.4 Query trees and efficiency

New candidates are generated by extending queries from the previous level. Any literals from the language can be added, as long as they agree with the modes, types and constraints of the language bias, and the whole association does not contain any part that is known to be infrequent. As each previous query can usually be extended by several literals, this leads naturally to a tree like structure of queries, where literals are nodes in the tree and children of a node are the possible extensions of the query up to that point. Each level in the tree corresponds to a level in the levelwise algorithm (or the length of an association). At the root of the tree is a single literal, which all queries contain. This is the "key" atom of WARMR.

Allowing common parts of queries to be collected up into a tree structure in this way provides several advantages. This was suggested by Luc Dehaspe (Dehaspe, 1998) (p104) as an improvement which could be made to WARMR. This is a compact way of representing queries, and it also means that counting can be done efficiently, since common subparts are counted just once. As the queries are first order, some thought is required to make sure the the various possibilities for variable bindings are consistent within a query, but this is feasible.

On running the algorithm and using time profiling, it became apparent that testing for subsumption accounted for most of the time taken. This is due to both the number of subsumption tests required, and the relatively expensive nature of this test. This time was then substantially alleviated and reduced by restricting the database literals to be tested - firstly to those with the correct predicate symbol, and secondly to those whose arguments match exactly with the constants in the literal of the query.

A further stage which can reduce the number of queries to be tested is to remove redundant queries. That is queries that are redundant in the sense that they are duplicates of or equivalent to existing queries, for example the same query with differently named variables. The test for equivalence of two queries Q_1 and Q_2 is if both Q_1 subsumes Q_2 and Q_2 subsumes Q_1 then they are equivalent, and one of them is unnecessary. At present this stage is carried out after PolyFARM has finished execution, but it could be added into the main body of the program.

6.3.5 Rules

PolyFARM is designed as an algorithm for finding frequent queries or associations and hence can be used to find association rules in the following manner:

> if $X \wedge Y$ and X are frequent associations
> then $X \rightarrow Y$ is a rule with confidence $support(X \wedge Y)/support(X)$.

If the user specifies which predicate is to be chosen as the head of the rule, then the query tree can be searched for a branch which ends in this predicate, and a rule can be produced from this branch.

However, these are not rules as such, and should instead be regarded as query extensions . The "rule" should correctly be written $X \rightarrow X \wedge Y$. Since queries are only existentially qualified, rather than universally as is the case with clauses, we cannot conclude the usual understanding of a rule. Luc Dehaspe (Dehaspe, 1998) gives the following example to illustrate:

$$\exists(buys(X, pizza) \wedge friend(X, Y)) \rightarrow \\ \exists(buys(X, pizza) \wedge friend(X, Y) \wedge buys(Y, coke))$$

which should be read:

> **if** a person X exists who buys *pizza* and has a friend Y
> **then** a person X exists who buys *pizza* and has a friend Y who buys *coke*

We shall henceforth call such implications "query extensions" to avoid confusion with rules, and adopt the notation of Luc Dehaspe in writing $X \rightsquigarrow Y$ to represent the extension of X by Y.

6.4 Results

Although such small data sets as are usually used in testing ILP systems do not require a distributed learner, we report results from PolyFARM on Michalski & Stepp's "Trains" data set from the UCI machine learning repository (Blake & Merz, 1998), just for comparison with other systems. We then show results for two large data sets from the yeast genome: predicted secondary structure data, and homology data.

6.4.1 Trains

The "Trains" dataset consists of a set of 10 trains, 5 of which are travelling east and 5 travelling west. The aim is to predict the direction of travel from a description of the cars in the train. There are between 3 and 5 cars per train, each car has associated properties such as number of wheels, position and load, and the loads in turn have properties such as their shape. Furthermore, one of the attributes of a car, "cshape" or the shape of the car, has a hierarchical value: some shapes are "opentop" and others are "closedtop".

This dataset illustrates many aspects of using a first order description of a problem. Variable numbers of facts can be used, and relations between them declared. The modes and types, shown in Figure 6.3 show the complexity and relationships between the data.

PolyFARM has no problems with this tiny dataset, and if minimum support is set to 0.5 and minimum confidence 1.0 we are presented with the following query extension results at level 5:

```
supp 5/10  conf 5/5
direction(_1,east) <---
       ccont(_1,_3),
       cshape(_3,_4),
       toptype(_4,closedtop),
       ln(_3,short).
```

That is, if a train has a short closedtop car then it's travelling east. This rule applies to all 5 of the eastbound trains and only these 5. No other query extensions are found at this level of support and confidence. Support must be set to 5/10 since this is the most we can expect if half of the dataset is one class and half is another. We chose confidence to be 100% in this case since it is a simple example and we are looking for rules which cover all cases exactly.

6.4.2 Predicted secondary structure (yeast)

Predicted secondary structure is highly important to functional genomics because the shape and structure of a gene's product can give clues to its function. The secondary structure information has a sequential aspect - for example, a gene might begin with a short alpha helix, followed by a long beta sheet and then another alpha helix. This spatial relationship between the components is important, and we wanted to use a relational data mining to extract features containing these relationships.

Data was collected about the predicted secondary structure of the gene products of *S. cerevisiae*. Prof (Ouali & King, 2000) was used to make the predictions.

```
mode(direction(+,constlist [east,west])).
mode(ccont(+,-)).
mode(ncar(+,constlist [3,4,5])).
mode(infront(+,+)).
mode(loc(+,constlist [1,2,3,4,5])).
mode(nwhl(+,constlist [2,3])).
mode(ln(+,constlist [short,long])).
mode(cshape(+,constlist [engine,openrect,slopetop,ushaped,
                         opentrap,hexagon,closedrect,dblopnrect,
                         ellipse,jaggedtop])).
mode(cshape(+,-)).
mode(npl(+,constlist [0,1,2,3])).
mode(lcont(+,-)).
mode(lshape(+,constlist [rectanglod,circlelod,hexagonlod,trianglod])).
mode(toptype(+,constlist [opentop,closedtop])).

type(ccont(Train,Car)).
type(ncar(Train,NumC)).
type(infront(Car,Car)).
type(loc(Car,Loc)).
type(nwhl(Car,NumW)).
type(ln(Car,Length)).
type(cshape(Car,Shape)).
type(npl(Car,NumP)).
type(lcont(Car,Load)).
type(lshape(Load,LoadT)).
type(direction(Train,Direction)).
type(toptype(Shape,TType)).
```

Figure 6.3: Mode and type declarations for the trains data set

The predictions were expressed as Datalog facts representing the lengths and relative positions of the alpha, beta and coil parts of the structure. The predictions also included the distributions of alpha, beta and coil as percentages. Table 6.1 shows the Datalog predicates that were used.

Each ORF had an average of 186 facts associated with it. 4,130 ORFs made up the database.

The Datalog facts were then mined with PolyFARM to extract frequently occurring patterns. Altogether, 19,628 frequently occurring patterns were discovered, where the minimum support threshold was 1/50, processing up to level 5 in the levelwise algorithm. The support threshold was determined by trial and error,

Predicate	Description
ss(Orf, Num, Type)	This Orf has a secondary structure prediction of type Type (alpha, beta or coil) at relative position Num. For example, ss(yal001c,3,alpha) would mean that the third prediction made for yal001c was alpha.
alpha_len(Num, AlphaLen)	The alpha prediction at position number Num was of length AlphaLen
beta_len(Num, BetaLen)	The beta prediction at position number Num was of length BetaLen
coil_len(Num, CoilLen)	The coil prediction at position number Num was of length CoilLen
alpha_dist(Orf, Percent)	The percentage of alphas for this ORF is Percent
beta_dist(Orf, Percent)	The percentage of betas for this ORF is Percent
coil_dist(Orf, Percent)	The percentage of coils for this ORF is Percent
nss(Num1, Num2, Type)	The prediction at position Num2 is of type Type (we used Num2 = Num1+1 ie Num1 and Num2 are neighbouring positions)

Table 6.1: Datalog facts collected for **struc** data.

in order to capture a large enough range of frequent patterns without obtaining too many that were infrequent. Deciding on a particular value for the minimum level of support is a known dilemma in association mining (Liu *et al.*, 1999), and there are no principled methods for its determination. Examples of the patterns found include:

$$ss(Orf, Num1, a), alpha\_len(Num1, b6 \ldots 10), alpha\_dist(Orf, b27.5 \ldots 36.2),$$
$$beta\_dist(Orf, b19.1 \ldots 29.1), coil\_dist(Orf, b45.4 \ldots 50.5).$$

This states that the ORF has a prediction of alpha with a length between 6 and 10, and that the alpha, beta and coil percentages are between 27.5 and 36.2%, between 19.1 and 29.1% and between 45.4 and 50.5% respectively.

$$ss(Orf, Num1, a), ss(Orf, Num2, a), alpha\_dist(Orf, b27.5 \ldots 36.2),$$
$$nss(Num1, Num3, c), nss(Num2, Num4, b).$$

This states that there are at least two alpha predictions for this ORF, one followed by a beta and the other followed by a coil, and that the distribution of alpha helices is between 27.5 and 36.2%.

More constraints would have been helpful for the structure data. Repeated predicates were allowed in associations for this data, in order to extract associa-

tions such as:

$$ss(Orf, X, a), ss(Orf, Z, a), coil\_dist(Orf, gte57.6), nss(X, Y, c), nss(Y, Z, a).$$

which represents an alpha helix at X followed by a coil at Y, followed by an alpha helix at Z, and the coil distribution is greater than or equal to 57.6%. This uses the *ss* and *nss* predicates more than once. However this meant that associations such as:

$$ss(Orf, X, a), ss(Orf, Y, a), ss(Orf, Z, a), alpha\_len(X, b1 \ldots 3), \\ alpha\_dist(Orf, b0.0 \ldots 17.9).$$

would also be found, where the variables X, Y and Z could possibly unify to the same position, and then the second two literals would be redundant. So allowing the user to specify such constraints as to prevent these variables unifying is a desirable future addition to PolyFARM.

6.4.3 Homology (yeast)

Data about homologous proteins is also informative. Homologous proteins are proteins that have evolved from the same ancestor at some point in time, and usually still share large percentages of their DNA composition. These proteins are likely to share common functions. We can search publicly available databases of known proteins to find such proteins that have sequences similar to our yeast genes.

Our homology data is the result of a PSI-BLAST search for each *S. cerevisiae* ORF against NRDB90. NRDB90 is a non-redundant protein database where proteins that share more than 90% similarity have been removed (Holm & Sander, 1998). This is created from the union of the SWISSPROT, SWISSNEW, TREMBL, TREMBLNEW, GenBank, PIR, WormPep and PDB databases. We used the version as of 4th January 2001 from http://www.ebi.ac.uk/~holm/nrdb90/, containing 260,000 non-duplicate sequences. PSI-BLAST (Altschul *et al.*, 1997) was used with the following parameters: "-e 10 -h 0.0005 -j 20". That is, a maximum of 20 iterations were run and the expectation value (e-value) cut-off was 10, since we required all similar sequences, even if only distantly similar. The e-value threshold for inclusion in the multipass model was 0.0005. The version of PSI-BLAST was BLASTP 2.0.12 [Apr-21-2000]. We also join on to nrdb90, the yeast genome itself, so that we can also discover similar sequences within the genome.

The sequences that are found by PSI-BLAST to have an e-value below the threshold are known as "hits". For each ORF, we extracted the SWISSPROT entries that were hits for the ORF. We used SWISSPROT version 39. Each

Fact	Description
sq_len(SPId, Len)	The sequence length of the SWISSPROT protein
mol_wt(SPId, MWt)	The molecular weight of the SWISSPROT protein
classification(SPId, Classfn)	The classification of the organism the SWISSPROT protein belonged to. This is part of a hierarchical species taxonomy. The top level of the hierarchy contains classes such as "bacteria" and "viruses" and the lower levels contain specific species such as "*E. coli*" and "*S. cerevisiae*".
keyword(SPId, KWord)	Any keywords listed for the SWISSPROT protein. Only keywords which could be directly ascertained from sequence were used. These were the following: transmembrane, inner_membrane, plasmid, repeat, outer_membrane, membrane.
db_ref(SPId, DBName)	The names of any databases that the SWISSPROT protein had references to. For example: PROSITE, EMBL, FlyBase, PDB.

Table 6.2: The facts which were extracted for each of the SWISSPROT entries that were PSI-BLAST hits for the yeast ORFS.

SWISSPROT entry was extracted from SWISSPROT and the facts shown in Table 6.2 were kept and translated into Datalog.

To these facts we also add a Datalog fact which contains the e-value for the hit, and Datalog facts containing the e-values for any hits that are part of the yeast genome itself. These are shown in Table 6.3.

Fact	Description
eval(Orf, SPId, EVal)	The e-value of the similarity between the ORF and the SWISSPROT protein
yeast_to_yeast(Orf, Orf, EVal)	The e-value between this ORF and another ORF in the yeast genome.

Table 6.3: The extra facts which were added to make up the **hom** data.

Numerical values were discretised by binning into 5 uniform-sized bins. Each ORF had an average of 5,082 facts associated with it. 4,252 ORFs made up the database.

The Datalog facts were then mined with PolyFARM to extract frequently occurring patterns. Altogether, 47,034 frequently occurring patterns were discovered, where the minimum support threshold was 1/20. Again, the support threshold was determined by trial and error, in order to capture a large enough

range of frequent patterns without obtaining too many that were infrequent. The search was stopped after level 3 since this was already a large number of patterns and clauses of 3 literals should capture enough complexity for further analysis.

Examples of patterns that were found include:

$$eval(Orf, SPID, b0.0 \ldots 1.0e{-}8), sq\_len(SPID, b16 \ldots 344), \\ classification(SPID, caenorhabditis).$$

This states that the ORF has a very close match to a SWISSPROT protein with a sequence length between 16 and 344 (short) which is from *Caenorhabditis*.

$$yeast\_to\_yeast(Orf, YeastORF, b3.3e{-}2 \ldots 0.73), \\ eval(Orf, SPID, b4.5e{-}2 \ldots 1.1), db\_ref(SPID, tuberculist).$$

This states that the ORF has a reasonably close match to another yeast ORF (with e-value between 0.033 and 0.73) and a reasonably close match (e-value between 0.045 and 1.1) to a SWISSPROT protein which has a reference in the tuberculist database.

6.5 Conclusion

We developed the PolyFARM algorithm to overcome scaling problems in data mining relational data. We applied it to the yeast predicted secondary structure and homology datasets. The application was successful and PolyFARM was able to handle all the databases.

PolyFARM is freely available for non-commercial use from http://www.aber.ac.uk/compsci/Research/bio/dss/polyfarm.

Chapter 7

Learning from hierarchies in functional genomics

7.1 Motivation

The desire to hierarchically classify objects in the natural world goes back to the days of Aristotle, and as early as the 18th century Linnaeus had catalogued eighteen thousand plant species. Today we have many biological classification systems, including catalogues of gene functions, cellular components, species, gene product interactions, anatomy and molecular structures.

This applies to yeast data in both the raw data for learning and in the classes we intend to learn. Therefore we needed to extend PolyFARM to deal with the hierarchical data and C4.5 to deal with hierarchical classes.

7.2 Hierarchical data - extending PolyFARM

7.2.1 Background

The homology dataset for yeast (this was described in more detail in Section 6.4.3) includes a species descriptor for the homologous genes. Species belong to a hierarchical taxonomy, and this taxonomy could be used to discover more general associations in the data. An example of part of this taxonomy is shown in Figure 7.1. In this example, "simplexvirus" is a child of "herpesviridae", which is a child of "viruses".

```
bacteria
  proteobacteria
    alpha_subdivision
       rickettsia
    beta_subdivision
       bordetella
       zoogloea
    gamma_subdivision
       escherichia
       shigella
viruses
  poxviridae
    orthopoxvirus
  herpesviridae
    simplexvirus
```

Figure 7.1: Example of part of species taxonomy for genes in SWISSPROT. Indentation shows isa relationships.

This can be dealt with by simply expanding out the hierarchy to give many separate facts for each ORF. For example, given the hierarchy in Figure 7.1, if an ORF was homologous to a protein "p10457" which was a simplexvirus protein, we could explicitly list all the following three facts about its classification:

```
classification(p10457,simplexvirus).
classification(p10457,herpesviridae).
classification(p10457,viruses).
```

In this way, all frequent associations could be found at any level of generality. However, with a deep hierarchy such as the one we have, this would mean that many extra facts needed to be associated with each ORF, and the species classification facts would become the major part of the database. Since most of this information could have been derived during processing this is wasteful of both disk space and memory to hold the data.

WARMR can deal with full Prolog predicates, meaning that the whole hierarchy can be specified as background knowledge. This would require defining parent/2 relationships and an ancestor/2 rule, which are classic Prolog textbook examples. Unfortunately, PolyFARM cannot yet deal with recursively defined predicates, and instead requires ground facts. The decision was made to add support for hierarchies more directly. A Prolog-like approach would only search these clauses by its traditional test-and-backtrack approach which can be very

slow when the list of predicates is large and there are many false candidates and dead ends. If instead the tree is encoded directly then it will be fast and simple to follow links between parent and child nodes. It will be interesting to add full support for the Prolog approach later and compare the two approaches.

7.2.2 Implementation in PolyFARM

In the settings file of PolyFARM, the user can now specify that an argument of a predicate contains a tree-structured value. The possible values will be provided in the background knowledge. For example, the following mode declaration:

```
mode(classification(+,consttree species)).
```

will declare that the "classification" predicate takes a tree-structured value as its second argument, whose values are from the "species" hierarchy. The "species" hierarchy is then given in the background knowledge. A small example follows (nested lists indicate the parent-child relationship):

```
hierarchy( species,
  [bacteria
     [salmonella,
      listeria,
      escherichia
     ],
   viruses
     [hepatitis_c-like_viruses,
      simplexvirus
     ]
  ]).
```

When a query is extended by the addition of a predicate with a tree-structured attribute, the candidate query to be tested against the database will contain the whole tree of possible values. When testing a query which contains a tree of values, a correct match should update all appropriate parts of the tree, including the ancestors of the value which actually matched.

When support of a query is counted, we count how many items match a query. An item may match a query in several ways. For example, the following query:

$$orf(X) \wedge similar(X, Y) \wedge classification(Y, consttree\ species).$$

would match against both ORFs in this database:

```
orf(1).
similar(1,p1111).
classification(p1111,salmonella)
similar(1,p9999)
classification(p9999,listeria).


orf(2).
similar(2,p5555).
classification(p5555,escherichia).
similar(2,p7777).
classification(p7777,hepatitis_c-like_viruses).
```

It would match against each ORF in two possible ways, and the support counts for salmonella, listeria, escherichia and hepatitis_c-like_viruses would all be incremented in the species hierarchy for this query. The counts would be propagated up to their parent species, so "bacteria" would have its support incremented for both ORFs, and "viruses" would have its support incremented for the second ORF. We must be careful when recording support counts that although the query matches ORF 1 in two different ways, from two different subspecies of bacteria, we only increase the support for the value "bacteria" by one, since we are only counting whether the query matches or not, and not how many times it matches.

After all matches have been determined, the tree can be pruned to remove all branches having less than the minimum support, and then the values remaining in the tree can be converted into ordinary arguments for ordinary literals. Figure 7.2 shows the support counts for the values of species argument of the previous query on the previous database.

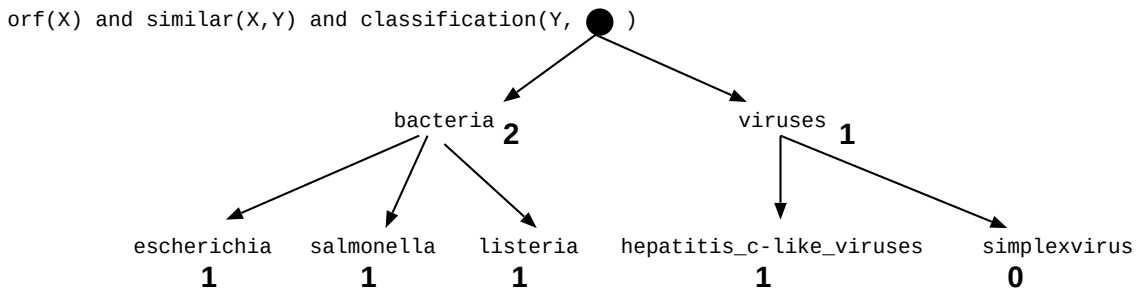


Figure 7.2: Support counts for the values of the species argument

If the minimum support threshold was 2 then the only query to remain after pruning would be:

$$orf(X) \wedge similar(X, Y) \wedge classification(Y, bacteria)$$

However, if the minimum support was 1 then the following six queries would remain after pruning:

$$orf(X) \wedge similar(X, Y) \wedge classification(Y, bacteria)$$
$$orf(X) \wedge similar(X, Y) \wedge classification(Y, escherichia)$$
$$orf(X) \wedge similar(X, Y) \wedge classification(Y, salmonella)$$
$$orf(X) \wedge similar(X, Y) \wedge classification(Y, listeria)$$
$$orf(X) \wedge similar(X, Y) \wedge classification(Y, viruses)$$
$$orf(X) \wedge similar(X, Y) \wedge classification(Y, hepatitis\_c\_like\_viruses)$$

When a query containing such a literal is to be further extended we need to remember that this value was once a tree-structured value. The subsumption test must be altered slightly to deal with tree-structured values. A constant in a query that came about from a tree-structured argument matches a constant in an example if the constants are equal or if the query constant is an ancestor of the example constant. Variables within a query that came about from a tree-structured argument match only if the bindings match exactly, as it would require deeper semantic knowledge of the predicates to allow otherwise. For example, one would expect the X in $classification(P, X) \wedge already\_sequenced(X)$ to be exactly the same, not bacteria in one literal and salmonella in the other.

If PolyFARM were being used to generate association rules or query extensions, then this method would also apply to hierarchies in the rule head. In this way, PolyFARM could also deal with learning hierarchical classes. However, association mining is not an efficient way to do rule learning, and PolyFARM, like WARMR, generates query extensions rather than real rules (see section 6.3.5). So in order to learn with hierarchical classes, we also extend C4.5.

7.3 Hierarchical classes - extending C4.5

7.3.1 Background

At the end of his book "C4.5: Programs for machine learning", Quinlan wrote a section on "Desirable Additions" which included "Structured attributes". In this, he suggested that it would be desirable to allow attributes to have a hierarchy of possible values. However, he did not suggest that the classes themselves might also

have hierarchically structured values. Almuallim *et al.* (1995; 1997) investigated his suggestion for hierarchically-valued attributes, both by ignoring/flattening the hierarchy, and by using the hierarchy directly to find the best value for the test on that attribute. They concluded that in their tests, the direct approach was more efficient and produced more general results. Kaufman and Michalski (1996) presents ideas for dealing with structured attributes, including the use of generalisation rules, and "anchor nodes" that allow the user to mark nodes "at preferable levels of abstraction" in the hierarchy. They apply their ideas in the INLEN-2 algorithm and discover simpler rules as a result. Little progress seems to have been made in recent years with hierarchically structured data. ILP-based algorithms can usually use hierarchical data by defining suitable recursive relations, though hierarchies have not been specifically investigated.

In using hierarchical classes there is little prior work. Most work in this area has been done in relation to classifying large volumes of text documents, for example to create Yahoo-like topic hierarchies. The classification algorithms used in these text processing applications tend to be either clustering or very simple statistical algorithms such as naïve Bayes, working on high volumes of data. Mitchell (1998) demonstrated that a hierarchical Bayesian classifier would have the same performance as a flat Bayesian classifier under certain assumptions: smoothing is not used to estimate the probabilities and the same features are used by different classifiers in the hierarchy. Work has been done on smoothing probabilities of items in low frequency classes by making use of their parent frequencies (McCallum *et al.*, 1998) and making more specific classifiers using different features at different places in the hierarchy (Koller & Sahami, 1997; Chakrabarti *et al.*, 1998; Mladenic & Grobelnik, 1998).

More recently a couple of papers have been published which look at the combined problem of both hierarchical classes and multiple labels. Wang *et al.* (2001) describe a method for classifying documents that is based on association rule mining. This method produces rules of the form $\{t_{i_1}, \ldots, t_{i_p}\} \rightarrow \{C_{i_1}, \ldots, C_{i_q}\}$, where the t_{i_j} are terms and the C_{i_j} are classes for a document i. A notion of similarity between class sets which takes into account the hierarchy is defined, and then certain rules are selected to construct the classifier.

Recently, Blockeel *et al.* (2002) have also designed an algorithm to tackle the problem of hierarchical multi-classification. They construct a type of decision tree called a "clustering tree" to do the classification, where the criteria for deciding on how to split a node is based on minimizing the intra-cluster variance of the data within a node. The distance measure used to calculate intra-cluster variance works on sets of class labels and takes into account both the class hierarchy and the multiple labels. They applied their method to our phenotype data (see Chapter 4) and obtained a small tree with just 2 tests. The top-level test was for calcofluor white, which provided the basis for our strongest rules too (c.f. our results in

section 4.6).

If we implement a learning algorithm which directly uses the hierarchy, instead of flattening it, then this should bring advantages because the dependencies between classes can be taken into account.

7.3.2 Implementation in C4.5

To adapt C4.5 to make use of a class hierarchy several modifications are needed:

- reading and storing the class hierarchy
- testing for membership of a class
- finding the best class or classes to represent a node
- performing entropy calculations

Reading and storing the class hierarchy

The class hierarchy is read from a file, along with the data values and attribute names. We require the user to unambiguously specify the hierarchical relationships by requesting the hierarchy already in tree format. The format uses indentation by spaces to show the parent and child relationships, with children indented more than their parents. Children can only have a single parent (unlike GeneOntology where multiple parents are allowed). Figure 7.3 shows part of our class hierarchy, and demonstrates the use of indentation to show relationships.

The functional class hierarchies we are using are wide and shallow. They have a high branching factor, but are only 4 levels deep. ORFs can belong to several classes at a time, at different levels in the hierarchy. We have three choices when recording class membership for each ORF:

1. Store only the most specific classes that the ORF belongs to in a variable sized array
2. Store all the classes that an ORF belongs to (including parent classes) in a variable sized array
3. Store all the classes that an ORF belongs to (including parent classes) in a fixed sized array which is as large as the total number of classes in the whole tree.

The first two options have the advantage of being economical with space. The last two options have the advantage of saving processing time computing the more general classes that the ORF also belongs to (we expect to need this calculation frequently). The third option has the added benefit of making simpler code.

```
metabolism
 amino acid metabolism
  amino acid biosynthesis
  amino acid degradation
  amino acid transport
 nitrogen and sulfur metabolism
  nitrogen and sulfur utilization
  regulation of nitrogen and sulphur utilization
energy
 glycolysis and gluconeogenesis
 respiration
 fermentation
cell cycle and DNA processing
 DNA processing
  DNA synthesis and replication
  DNA repair
 cell cycle
  meiosis
  chromosome condensation
```

Figure 7.3: Format for reading the class hierarchy into C4.5. Indentation shows the parent-child (isa) relationship. In this example, "nitrogen and sulfur metabolism" is a child of "metabolism" and "DNA processing" is a child of "cell cycle and DNA processing".

We chose the third option and stored a fixed length Boolean array with each ORF, representing the classes it belongs to by true values in the appropriate elements of the array. Given the high branching factor and short depth of the tree, the space overhead in storing all classes instead of the most specific classes is very slight. Also, in our functional genomics datasets, the number of classes in total will be less than 300, whereas the number of attribute values for each ORF can be thousands. So the memory overhead will be relatively small. Explicitly representing all possible classes means faster processing time.

Since the hierarchy is now flattened into an array for each data item's classes, we must also store the hierarchy itself, so that parent/child relationships can be reconstructed and used to index into the array. This was achieved by a data structure of linked structs. With a multiway-branching tree such as this we must explicitly represent links to parent, siblings and children. The following struct, which represents a node in the hierarchy, contains pointers to a parent, sibling and child, the position of this class in the array and the total number of descendents of this class.

```
struct Classtree
{
  char name[NAMELENGTH];
  struct Classtree * parent;
  struct Classtree * sibling;
  struct Classtree * child;
  int arraypos;             /* position in array of this class */
  int numdescendants;       /* number of descendents, for entropy */
};
```

We also need a reverse index, from array position to tree node, so that info can be extracted quickly from the tree.

Tests for membership of a class

Testing a data item for membership of a class is now trivial, due to our class representation: simply check if the appropriate array element is set to "true" or not. Membership of parent classes was calculated once at the start, and is now explicit.

When doing calculations which involve looking at a specific level in the hierarchy, we can use a Boolean mask to hide classes belonging to other levels. Since the class array consists of Booleans, simply AND-ing a Boolean array mask with the class array will show the classes of interest.

Finding the best class or classes to represent a node

Nodes in the decision tree are labelled with the classes which best represent the data in each node. These classes are the classification which is to be made by following a path from the root down to that node. Since we are still dealing with the multi-label problem (see Chapter 4), there may be several classes used to label a node in the tree. When dealing with the class hierarchy we could choose to label the node with all appropriate classes from most specific to most general, or just the most specific classes. For example, should we say that all ORFs at a node are involved in both "respiration" and "fermentation", or in "respiration", "fermentation" and their parent class "energy"? We chose to find only the most specific classes, since these will be the most interesting and useful when making predictions for ORFs of unknown function.

Since frequency of class representation is monotonic with respect to moving down the hierarchy (a child class will never be more frequent than its parent) we can start by simply finding the most frequent set of classes represented in the data at the top (most general) level of the hierarchy. Given this set of classes S and its frequency F, we know that S is the most frequent pattern of classes represented in the data. Each class in S which has children could potentially be specialised

and replaced by one or more of its children. We refine S by searching down the hierarchy for the best specialisations of each of the classes, while still requiring frequency F for the specialised set.

Entropy calculations

To deal with hierarchies the entropy calculations again need to be modified. The basic entropy calculations developed for multi-label learning (see Section 4.4) still apply, but we also need to consider the differences between levels of the hierarchy. If we have a data set where all items are of one class (say "energy") then the C4.5 algorithm would normally terminate. However, if this set can be partitioned further into two subsets, one of elements in the child class "respiration" and one of elements in the child class "fermentation", then we would like to continue partitioning the data.

The original entropy calculation was equal to 0 for both of these cases. But we know that more specific classes are more informative than the more general classes. A partition into two child classes is more informative than a single label of the parent class. This can be used when calculating entropy (which is after all a measurement of the uncertainty, or lack of information). We can say that reporting just the result "energy" when energy has two subclasses is as if we had reported "fermentation or respiration: we don't know which". So reporting a more general class should cost as much as reporting all of its subclasses. "energy" is a more uncertain answer than "respiration". With this in mind, we have a new calculation for entropy:

$$entropy = -\sum_{i=1}^{N} ((p(c_i)\log p(c_i)) + (q(c_i)\log q(c_i)) - \alpha(c_i)\log treesize(c_i))$$

where

$$\begin{aligned} p(c_i) &= \text{probability (relative frequency) of class } c_i \\ q(c_i) &= 1 - p(c_i) = \text{probability of not being member of class } c_i \\ treesize(c_i) &= 1 + \text{number of descendant classes of class } c_i \\ &\quad (1 \text{ is added to represent } c_i \text{ itself}) \\ \alpha(c_i) &= 0, \text{if } p(c_i) = 0 \\ &\quad \text{a user defined constant (default=1) otherwise.} \end{aligned}$$

The entropy is now composed of two parts:

- $(p(c_i)\log p(c_i)) + (q(c_i)\log q(c_i))$ which is the uncertainty in the choice of class labels

- "$\log treesize(c_i)$" which is the uncertainty in the specificity of the class labels, and represents transmitting the size of the class hierarchy under the class in question.

α is primarily a constant, decided by the user, which allows a weighting to be given to the specificity part of the formula. The default value is 1, which means that the uncertainty in the choice and uncertainty in the specificity have equal weighting. Increasing the value of α would mean that the user was much more interested in having specific classes reported at the expense of homogeneity, and decreasing its value would favour more general classes if they made the nodes homogeneous. α is set to 0 if the class probability is zero, since there is no need to transmit information about its treesize if this class is not used.

Chapter 8

Data combination and function prediction

8.1 Introduction

During the course of this work we have collected the datasets listed in Table 8.1. We also list in this table the dataset "expr" formed by using all microarray datasets together. These datasets will henceforth be known as "individual" datasets (as opposed to compound datasets, which shall be constructed later). These datasets will be described in more detail in the following sections.

We used these datasets, and various combinations of these datasets, to develop rules which discriminate between the functional classes. We analysed these rules for accuracy and biological significance, and used them to make predictions for genes of currently unknown function.

The rule learning program was C4.5, modified to use multiple labels as described in Chapter 4 and hierarchical classes as described in Chapter 7. We also learn each level of the class hierarchy independently so the hierarchical learning can be compared with non-hierarchical learning.

We used the standard 3-way split of data for measuring accuracy of our results. The data was split into 3 parts: training, validation and test. The training data was used to create the rules. The validation data was used to select the best rules. The test data was used to estimate the accuracy of the selected rules on unseen data. All three parts will be independent. Figure 8.1 shows how the parts are used and their relative sizes.

Name	Description
seq	Data consisting only of attributes that can be calculated from sequence alone (for example amino acid ratios, sequence length and molecular weight)
pheno	Data from phenotype growth experiments
struc	Data from secondary structure prediction. Boolean attributes were constructed from the first order patterns mined by PolyFARM.
hom	Data from the results of PSI-BLAST searches of NRPROT. Boolean attributes were constructed from the first order patterns mined by PolyFARM.
cellcycle	Microarray data from Spellman *et al.* (1998)
church	Microarray data from Roth *et al.* (1998)
derisi	Microarray data from DeRisi *et al.* (1997)
eisen	Microarray data from Eisen *et al.* (1998)
gasch1	Microarray data from Gasch *et al.* (2000)
gasch2	Microarray data from Gasch *et al.* (2001)
spo	Microarray data from Chu *et al.* (1998)
expr	All microarray datasets concatenated together.

Table 8.1: Individual datasets

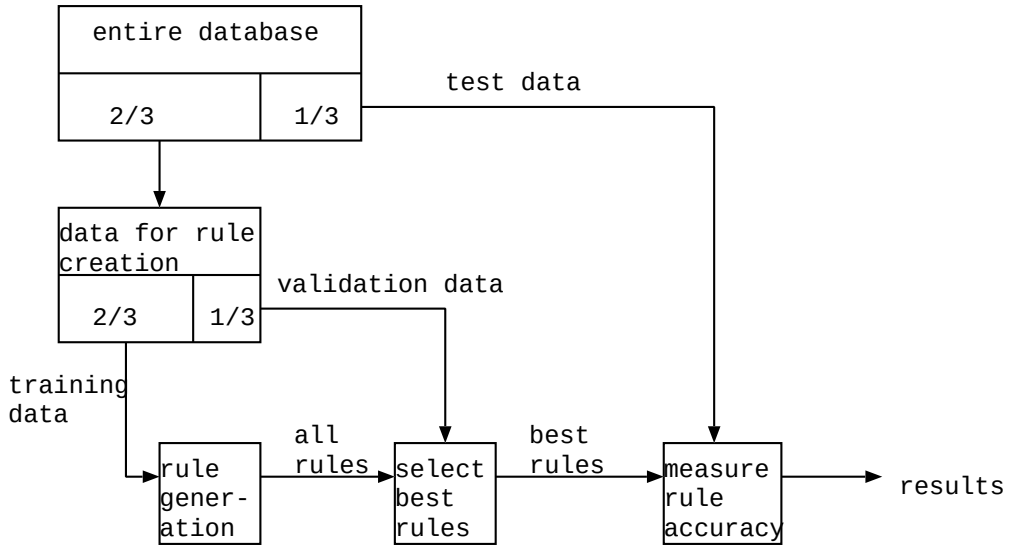


Figure 8.1: The data was split into 3 parts, training data, validation data and test data. Training data was used for rule generation, validation data for selecting the best rules and test data for measuring rule accuracy. All three parts were independent.

8.2 Validation

All tables are given for both the whole rulesets and the rulesets after validation has been applied (i.e. just the significant rules). The validation was applied by keeping only the rules which were shown to be statistically significant on the validation data set. See Figure 8.1 for a diagram of how the validation data set relates to the training and test data. Statistical significance was calculated by using the hypergeometric distribution with an α value of 0.05 and a Bonferroni correction.

The hypergeometric distribution is the distribution which occurs when we take a sample without replacement from a population which contains two types of elements, and we want to see how many of one of the types of elements we would expect to find in our sample. The probability of obtaining exactly k elements is given by the following equation:

$$P(X = k) = \frac{C(R, k) * C(N - R, n - k)}{C(N, n)} \tag{8.1}$$

for $k = max(0, n - (N - R)), \ldots, min(n, R)$, where n = number in sample, k = number in our sample which are the class of interest, N = total population size and R = total number of elements of class of interest in whole population.

The Bonferroni correction adjusts for the situation where we are looking for a statistically significant result in many tests. Salzberg (1997), among others, describes the use of this correction and the problems in evaluating and comparing classifiers. We are likely to find some statistically significant result just by chance if the number of tests is large. The Bonferroni correction is very simple, and just adjusts the α value down to compensate for the number of tests. There is much debate over the drawbacks of using this correction, since it will penalise too harshly if the tests are correlated (Feise, 2002; Perneger, 1998; Bender & Lange, 1999). However, since we are looking for only the most accurate and general rules, then losing rules that should have been valid is better than keeping insignificant rules (we prefer a type II error).

8.3 Individual datasets

In this section the individual data sets are described in more detail.

8.3.1 seq

The **seq** data was collected from a variety of sources. It is mostly numerical attribute-value data. The attributes are as shown in Table 8.2.

8.3.2 pheno

The **pheno** data is exactly as described in Chapter 4. This data represents phenotypic growth experiments on knockout yeast mutants.

8.3.3 struc

The **struc** data is data about the predicted secondary structure of the protein. The data is exactly as described in Section 6.4.2.

The data was then mined with PolyFARM (see Chapter 6) to extract frequently occurring patterns. These patterns are then converted into boolean attributes for each ORF: a 1 indicates that this pattern is present in this ORF and a 0 indicates that this pattern is absent. Altogether 19,628 frequently occurring patterns were discovered, so 19,628 boolean attributes exist in this dataset.

8.3.4 hom

The **hom** data is the result of a PSI-BLAST search (homology search) for each ORF. The data is exactly as described in Section 6.4.3.

The data was then mined with PolyFARM (see Chapter 6) to extract frequently occurring patterns. These patterns are then converted into boolean attributes for each ORF: a 1 indicates that this pattern is present in this ORF and a 0 indicates that this pattern is absent. Altogether 47,034 frequently occurring patterns were discovered, so 47,034 boolean attributes exist in this dataset.

8.3.5 cellcycle

This is microarray data from Spellman *et al.* (1998). This data consisted of 77 real-valued attributes which came from 4 time-series experiments. The data was obtained from http://genome-www.stanford.edu/cellcycle/data/rawdata/.

Attribute	Type	Description
aa_rat_X	real	Percentage of amino acid X in the protein
seq_len	integer	Length of the protein sequence
aa_rat_pair_X_Y	real	Percentage of the pair of amino acids X and Y consecutively in the protein
mol_wt	integer	Molecular weight of the protein
theo_pI	real	Theoretical pI (ioselectric point)
atomic_comp_X	real	Atomic composition of X where X is c (carbon), o (oxygen), n (nitrogen), s (sulphur) or h (hydrogen)
aliphatic_index	real	The aliphatic index
hydro	real	Grand average of hydropathicity
strand	'w' or 'c'	The DNA strand on which the ORF lies
position	integer	Number of exons (how many start positions are there in its coordinates list).
cai	real	Codon adaption index: calculated according to Sharp and Li (1987)
motifs	integer	Number of motifs: according to PROSITE dictionary release 13 of Nov. 1995 (Bairoch *et al.*, 1996)
transmembraneSpans	integer	Number of transmembrane spans: calculation follows Klein *et al.* (1985) using the ALOM program. P:I threshold value of 0.1 is used for ORF products which have at least only one transmembrane span. P:I threshold value of 0.15 is used for all TM-calculated proteins. (Goffeau *et al.*, 1993)
chromosome	1..16, mit	Chromosome number for this ORF

Table 8.2: **seq** attributes. Attributes in the top section of this table are calculated directly. Attributes in the middle section were calculated by Expasy's ProtParam tool. Attributes at the bottom are from MIPS' chromosome tables (dated 20/10/00 on the MIPS web site).

8.3.6 church

This is microarray data from the Church lab by Roth *et al.* (1998). It consists of 27 mostly real-valued attributes. The data was obtained from http://arep.med.harvard.edu/mrnadata/expression.html.

8.3.7 derisi

This is microarray data from DeRisi *et al.* (1997) investigating the diauxic shift. It consists of 63 real-valued attributes. The data was obtained from http://cmgm.stanford.edu/pbrown/explore/additional.html.

8.3.8 eisen

This is microarray data from Eisen *et al.* (1998). It consists of 79 real-valued attributes. This dataset is a composite dataset, consisting of data from the 4 cellcycle experiments, the sporulation experiments, the derisi experiments and some additional experiments on heat/cold shock. The data was obtained from http://rana.stanford.edu/clustering/.

8.3.9 gasch1

This is microarray data from Gasch *et al.* (2000). It consists of 173 real-valued attributes. The data was obtained from http://genome-www.stanford.edu/yeast_stress/data/rawdata/complete_dataset.txt.

8.3.10 gasch2

This is microarray data from Gasch *et al.* (2001). It consists of 52 real-valued attributes. The data was obtained from http://genome-www.stanford.edu/Mec1/data/DNAcomplete_dataset/DNAcomplete_dataset.cdt.

8.3.11 spo

This is microarray data from Chu *et al.* (1998). It consists of 80 mostly real-valued attributes. The data was obtained from http://cmgm.stanford.edu/pbrown/sporulation/additional/.

8.3.12 expr

This dataset consists of the direct concatenation of the all the microarray datasets described above. This means there is some duplication in the data since the **eisen**

dataset already contains some of the others. Also, since the different datasets cover different ORFs, there will be some ORFs which have missing values. These are represented in C4.5 with the "?" character.

8.4 Functional classes

The functional classification scheme that was used in all these experiments was from MIPS[1] and was taken on 24/4/02. Four levels of this hierarchy were used. The top level has 19 classes, including the classes "UNCLASSIFIED PROTEINS" and "CLASSIFICATION NOT YET CLEAR-CUT". Table 8.3 shows all top level classes. Table 8.4 shows all level 2 classes which are represented in the results which follow in this chapter (since the results are given by class number, this table can be used to look up the actual name of the class).

[1] http://mips.gsf.de/proj/yeast/catalogues/funcat/

ID number	Name
1,0,0,0	METABOLISM
2,0,0,0	ENERGY
3,0,0,0	CELL CYCLE AND DNA PROCESSING
4,0,0,0	TRANSCRIPTION
5,0,0,0	PROTEIN SYNTHESIS
6,0,0,0	PROTEIN FATE (folding, modification, destination)
8,0,0,0	CELLULAR TRANSPORT AND TRANSPORT MECHANISMS
10,0,0,0	CELLULAR COMMUNICATION/SIGNAL TRANSDUCTION MECHANISM
11,0,0,0	CELL RESCUE, DEFENSE AND VIRULENCE
13,0,0,0	REGULATION OF / INTERACTION WITH CELLULAR ENVIRONMENT
14,0,0,0	CELL FATE
29,0,0,0	TRANSPOSABLE ELEMENTS, VIRAL AND PLASMID PROTEINS
30,0,0,0	CONTROL OF CELLULAR ORGANIZATION
40,0,0,0	SUBCELLULAR LOCALISATION
62,0,0,0	PROTEIN ACTIVITY REGULATION
63,0,0,0	PROTEIN WITH BINDING FUNCTION OR COFACTOR REQUIREMENT (structural or catalytic)
67,0,0,0	TRANSPORT FACILITATION
98,0,0,0	CLASSIFICATION NOT YET CLEAR-CUT
99,0,0,0	UNCLASSIFIED PROTEINS

Table 8.3: Top level classes from MIPS classification scheme, 24/2/02.

ID number	Name
1,1,0,0	amino acid metabolism
1,2,0,0	nitrogen and sulfur metabolism
1,3,0,0	nucleotide metabolism
1,5,0,0	C-compound and carbohydrate metabolism
2,13,0,0	respiration
3,1,0,0	DNA processing
3,3,0,0	cell cycle
4,1,0,0	rRNA transcription
4,5,0,0	mRNA transcription
5,1,0,0	ribosome biogenesis
5,10,0,0	aminoacyl-tRNA-synthetases
6,13,0,0	proteolytic degradation
8,4,0,0	mitochondrial transport
8,19,0,0	cellular import
11,7,0,0	detoxification
30,1,0,0	cell wall
40,2,0,0	plasma membrane
40,3,0,0	cytoplasm
40,7,0,0	endoplasmic reticulum
40,10,0,0	nucleus
40,16,0,0	mitochondrion
67,10,0,0	amino-acid transporters
67,28,0,0	drug transporters
67,50,0,0	transport mechanism

Table 8.4: Level 2 classes from MIPS classification scheme, 24/2/02. Only a subset of classes are shown (just those that are represented in the following results tables).

8.5 Individual dataset results

The following tables show the average accuracies of the rulesets, the class by class accuracies, the coverage, the number of predictions made for ORFs of unknown function, the number of rules in each ruleset and the number of rules which predict more than one homology class or a new homology class.

8.5.1 Accuracy

Average accuracy of the whole rulesets and the rulesets after validation are shown in Tables 8.5 and 8.6 respectively. The rulesets after validation have had all non-significant rules removed.

The average accuracies of the validated rulesets in Table 8.6 range between 75% and 39% on level 1, dropping on the lower levels to 0% at level 4 where data is sparse. 75-39% is very good when compared to the *a priori* class probabilities. Tables 8.7, 8.8 and 8.9 give class by class breakdowns for the higher levels, along with *a priori* class probabilities for comparison. Some classes have high *a priori* probabilities (class 40,0,0,0 "subcellular localisation" has a prior of 57%), but most *a priori* probabilities are less than 20%. Where the table entry is blank we have no rules that predict this class. Some classes are obviously predicted better than others, and some types of data predict certain classes better than others. Class 5,0,0,0 - "protein synthesis" (and in particular its subclass 5,1,0,0 - "ribosome biogenesis") is consistently predicted very well by most datasets, especially the expression datasets. Its *a priori* probability is just 9% but the rulesets are between 55% and 93% accurate on this class. The **seq** data is a good predictor of 29,0,0,0 - "transposable elements, viral and plasmid proteins" and the **pheno** data of 3,0,0,0 - "cell cycle and DNA processing", and 30,1,0,0 - "cell wall" (as shown in Chapter 4). **hom** data picks out 8,4,0,0 - "mitochondrial transport", 6,0,0,0 - "protein fate" and 67,0,0,0 - "transport facilitation".

The **hom** dataset produces rules for many of the classes, showing a broad spread in its capabilities. The **gasch1** dataset also applies to many classes, however the rules are of much lower accuracy. The diversity of the classes represented reflects the nature of the **gasch1** dataset, which was produced by measuring the effects of a diverse range of conditions, including temperature shock, hydrogen peroxide, menadione, hyper- and hypo-osmotic shock, amino acid starvation and nitrogen source depletion. Other expression data sets were generally created by measuring one specific effect such as the progression of the cell cycle.

The accuracy of the combined expression dataset is not higher than the individual expression sets, and in fact it is usually lower. This is surprising since more information is available in the combined dataset.

The accuracy of the hierarchical version of C4.5 is sometimes better and sometimes worse than the accuracy of standard C4.5 on the individual levels. This is

disappointing, as we would expect that given more information the results should be consistently better. The hierarchical C4.5 does not find as many rules as would be found by learning all the levels individually. Also, those rules that it does produce are different to the rules produced individually. This is to be expected, as the criteria for choosing nodes in the decision tree are slightly different, and a different amount of information is available.

	level				
datatype	1	2	3	4	all
seq	54	44	50	25	60
pheno	52	32	17	31	40
struc	53	48	38	20	58
hom	57	36	38	20	58
cellcycle	53	33	23	31	50
church	59	33	18	31	47
derisi	56	40	18	35	58
eisen	73	39	28	27	61
gasch1	52	43	29	43	48
gasch2	52	61	32	50	55
spo	56	42	25	57	54
expr	54	37	32	38	58

Table 8.5: **Accuracy**: Average accuracy (percentages) on the test data of each ruleset produced by the individual datasets. All generated rules are included. Level "all" indicates the results of the hierarchical version of C4.5, which had classes from all levels in its training data.

	level				
datatype	1	2	3	4	all
seq	55	55	33	0	71
pheno	75	40	7	0	68
struc	49	43	0	0	58
hom	65	38	69	20	55
cellcycle	63	33	21	0	54
church	75	43	0	0	53
derisi	64	51	0	0	61
eisen	63	40	28	0	48
gasch1	39	46	44	75	38
gasch2	44	66	40	0	60
spo	43	63	0	0	46
expr	42	37	35	0	75

Table 8.6: **Accuracy**: Average accuracy (percentages) on the test data of each VALIDATED ruleset produced by the individual datasets. Only rules which were statistically significant on the validation set are included. Level "all" indicates the results of the hierarchical version of C4.5, which had classes from all levels in its training data. Significance was calculated by the hypergeometric distribution, with alpha=0.05 and Bonferroni correction.

class	prior	seq	pheno	struc	hom	cellcycle	church	derisi	eisen	gasch1	gasch2	spo	expr
1,0,0,0	27	52			65				53	41			47
2,0,0,0	6				20				47	29			38
3,0,0,0	16		75			67				30	24	28	
4,0,0,0	20	33											33
5,0,0,0	9	72			86	56	75	71	91	81	74	81	93
6,0,0,0	15				100					35			
8,0,0,0	12				42								
29,0,0,0	3	85		33	63								
30,0,0,0	5		75										
40,0,0,0	57	61	75			64		64					
67,0,0,0	8	42		49	77					24			

Table 8.7: Class by class accuracies (percentages) for individual datasets, level 1, VALIDATED rules only.

class	prior	seq	pheno	struc	hom	cellcycle	church	derisi	eisen	gasch1	gasch2	spo	expr
1,1,0,0	5									50			60
1,2,0,0	2									25			
1,3,0,0	4				46								
2,13,0,0	2							24	45	29			50
3,1,0,0	7					42			50				
3,3,0,0	12											50	
4,1,0,0	3									29	17		
4,5,0,0	15				56				25				30
5,1,0,0	5				81	66	33	77	73	84	78	84	88
5,10,0,0	1	80			83								
6,13,0,0	4								67	54	71		42
8,4,0,0	2			80	100								
8,19,0,0	3			50									
11,7,0,0	3			50									
30,1,0,0	3		83										
40,2,0,0	4	60			28								
40,3,0,0	15	80			51	58	57	54	79	78	73	57	77
40,7,0,0	4								56		80		
40,10,0,0	20		25	35	30	28			35	32			28
40,16,0,0	9	15	31			33	30	32	33	62	52		33
67,10,0,0	1				31								
67,28,0,0	1			50	18								
67,50,0,0	2				100								

Table 8.8: Class by class accuracies (percentages) for individual datasets, level 2, VALIDATED rules only.

class	prior	seq	pheno	struc	hom	cellcycle	church	derisi	eisen	gasch1	gasch2	spo	expr
1,0,0,0	27	48					36			50			
1,5,1,0	7				60								
2,13,0,0	2							63					
2,16,0,0	1										0		
3,0,0,0	16												44
3,1,3,0	3		67										
4,0,0,0	20					34				29			
5,0,0,0	9	78				33	55	58	64	78	56	56	87
5,1,0,0	5	77			78	64	61	54	56	83	100	79	78
6,13,1,0	3									20			
8,4,0,0	2			73	100								
29,0,0,0	3	81			55								
30,1,0,0	3		69										
40,0,0,0	57					61	65				64		
40,2,0,0	4	14											
40,3,0,0	14	81			35	57		64	55	43	56	47	80
40,10,0,0	20								39	25		26	
40,16,0,0	9								38		29		
67,0,0,0	8	78		55	88								
67,28,0,0	1			44									

Table 8.9: Class by class accuracies (percentages) for individual datasets, all levels (hierarchical learning), VALIDATED rules only.

8.5.2 Coverage

Coverage varies widely depending on the type of data (see Tables 8.10 and 8.11 for all rules and validated rules respectively). The coverage also varies widely at the different levels of classification. At level 1 the **seq** dataset gives the best coverage, whereas at level 2 the best coverage is provided by the **eisen** dataset. Using hierarchical learning, the dataset for best coverage is different again, this time **cellcycle**. Each dataset will have its strengths in the prediction of different classes and this is highlighted by the coverage figures.

Coverage and accuracy are related differently for each dataset. In general our results show better accuracy than coverage: this is due to our validation procedure where we select rules based on their accuracy. We are more interested in making correct predictions than in making many predictions. The spread of coverage versus accuracy for each of the individual datasets can be seen in Figure 8.2 for level 1, Figure 8.3 for level 2 and Figure 8.4 for hierarchical learning, all levels. A good spread of values exists for each level.

datatype	level				
	1	2	3	4	all
seq	79.36 (1065)	18.95 (248)	3.03 (27)	8.85 (33)	60.13 (807)
pheno	84.19 (490)	29.04 (169)	16.41 (65)	48.82 (83)	64.09 (373)
struc	75.23 (990)	5.83 (76)	3.27 (29)	1.35 (5)	72.57 (955)
hom	66.46 (876)	38.37 (493)	8.63 (76)	1.36 (5)	56.53 (745)
cellcycle	75.62 (971)	43.99 (564)	30.08 (265)	35.79 (131)	84.50 (1085)
church	61.92 (795)	19.34 (248)	5.57 (49)	28.42 (104)	68.54 (880)
derisi	69.36 (876)	28.47 (359)	3.10 (27)	13.54 (49)	13.86 (175)
eisen	77.78 (651)	57.47 (481)	34.77 (202)	39.67 (96)	87.34 (731)
gasch1	84.85 (1092)	39.84 (512)	11.26 (99)	25.68 (94)	86.17 (1109)
gasch2	89.34 (1156)	13.62 (176)	9.28 (82)	7.63 (28)	67.23 (870)
spo	65.43 (827)	20.60 (260)	4.58 (40)	3.86 (14)	66.22 (837)
expr	95.75 (1239)	43.73 (565)	12.22 (108)	29.16 (107)	99.30 (1285)

Table 8.10: **Coverage**: Test set coverage of each ruleset produced by the individual datasets. Figures are given in percentages with actual numbers of ORFs in brackets. All generated rules are included. Level "all" indicates the results of the hierarchical version of C4.5, which had classes from all levels in its training data.

datatype	level				
	1	2	3	4	all
seq	79.28 (1064)	10.01 (131)	1.68 (15)	0.00 (0)	14.16 (190)
pheno	12.20 (71)	14.78 (86)	7.58 (30)	0.00 (0)	3.26 (19)
struc	7.60 (100)	5.07 (66)	0.00 (0)	0.00 (0)	2.05 (27)
hom	17.00 (224)	36.73 (472)	2.95 (26)	1.36 (5)	12.06 (159)
cellcycle	51.64 (663)	37.68 (483)	23.04 (203)	0.00 (0)	71.34 (916)
church	2.49 (32)	10.06 (129)	0.00 (0)	0.00 (0)	58.64 (753)
derisi	60.33 (762)	12.93 (163)	0.00 (0)	0.00 (0)	8.39 (106)
eisen	17.68 (148)	51.37 (430)	28.74 (167)	0.00 (0)	37.63 (315)
gasch1	47.55 (612)	33.39 (429)	3.07 (27)	1.09 (4)	47.24 (608)
gasch2	13.68 (177)	11.07 (143)	0.57 (5)	0.00 (0)	64.06 (829)
spo	9.97 (126)	8.32 (105)	0.00 (0)	0.00 (0)	12.82 (162)
expr	37.94 (491)	43.11 (557)	7.35 (65)	0.00 (0)	5.56 (72)

Table 8.11: **Coverage**: Test set coverage of each VALIDATED ruleset produced by the individual datasets. Figures are given in percentages with actual numbers of ORFs in brackets. Only rules which were statistically significant on the validation set are included. Level "all" indicates the results of the hierarchical version of C4.5, which had classes from all levels in its training data. Significance was calculated by the hypergeometric distribution, with alpha=0.05 and Bonferroni correction.

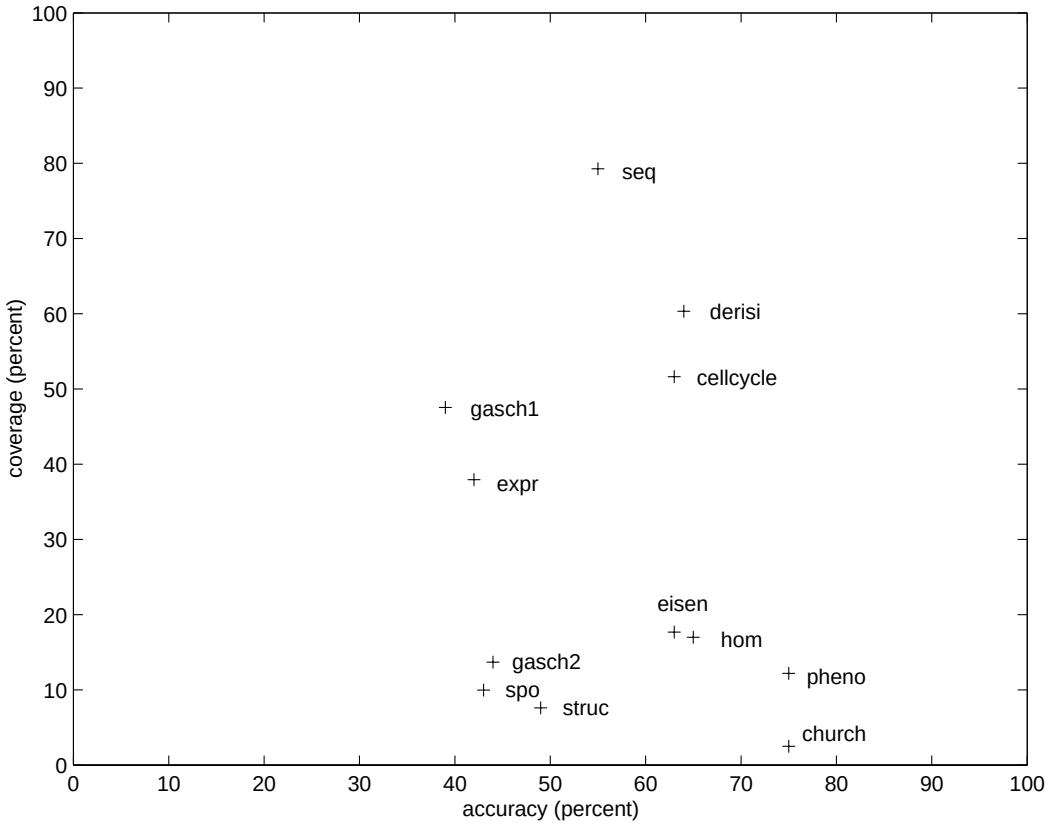


Figure 8.2: Individual VALIDATED datasets at level 1: coverage versus accuracy

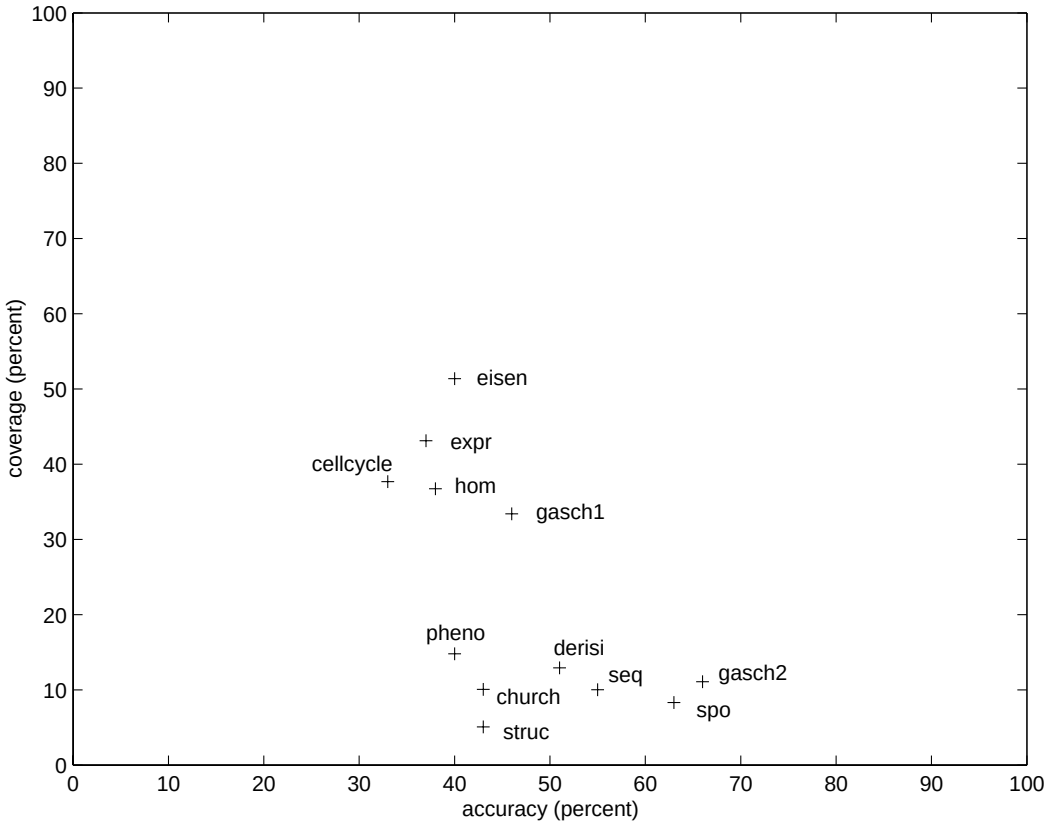


Figure 8.3: Individual VALIDATED datasets at level 2: coverage versus accuracy

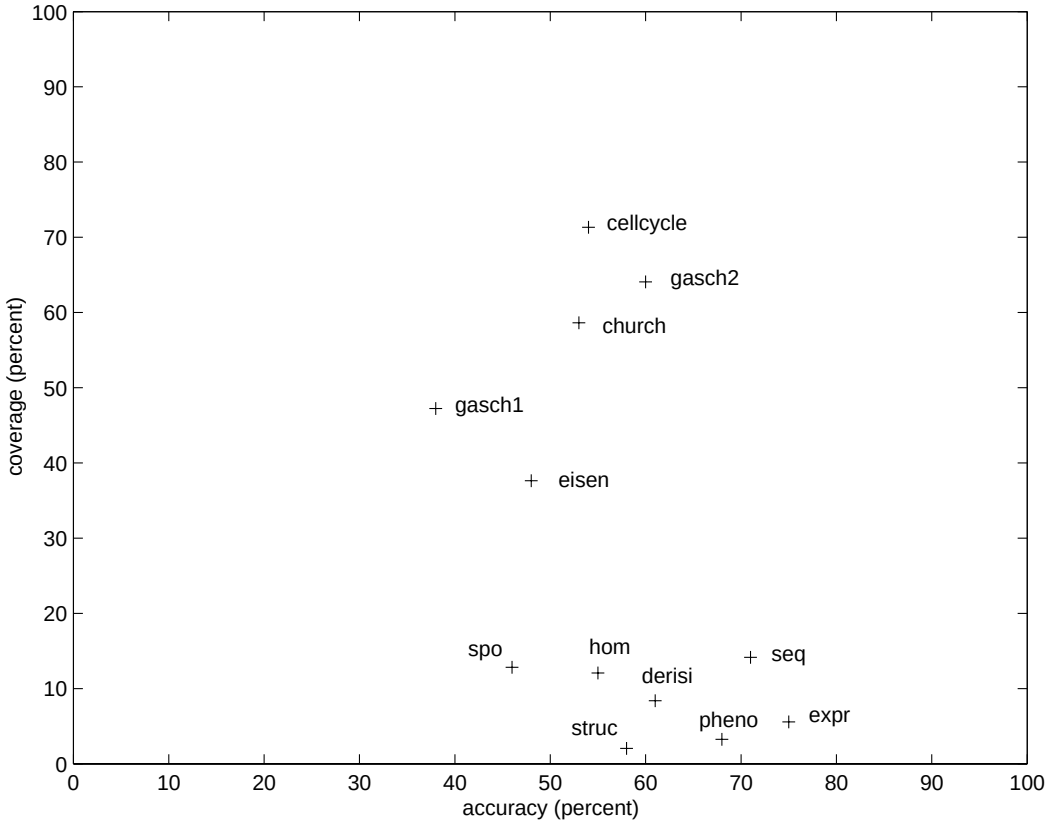


Figure 8.4: Individual VALIDATED datasets in hierarchical learning (level "all"): coverage versus accuracy

8.5.3 Predictions

The numbers of predictions can be found in Tables 8.12 and 8.13 (for all rules and validated rules respectively). The **seq** data makes the most predictions at level 1 with 1646 ORFs assigned some function. However this drops sharply, with **seq** predicting only 39 ORFs at level 2. The expression data sets also make large numbers of predictions, and the combined expression data makes a huge number of predictions.

	level				
datatype	1	2	3	4	all
seq	2294 (1672)	285 (259)	40 (40)	334 (314)	1589 (1369)
pheno	796 (689)	169 (129)	92 (91)	384 (377)	484 (438)
struc	2102 (1841)	147 (105)	18 (18)	27 (27)	1846 (1736)
hom	633 (471)	373 (305)	30 (27)	13 (13)	1171 (1149)
cellcycle	1954 (1576)	930 (882)	583 (567)	907 (787)	2710 (1816)
church	1303 (1243)	312 (274)	65 (63)	589 (563)	1870 (1365)
derisi	1611 (1369)	520 (479)	61 (54)	452 (436)	208 (130)
eisen	35 (30)	34 (26)	16 (16)	18 (17)	53 (33)
gasch1	2367 (1742)	955 (841)	175 (163)	505 (502)	3183 (1999)
gasch2	2744 (2135)	347 (250)	223 (222)	257 (254)	1991 (1583)
spo	1397 (1265)	376 (333)	77 (77)	45 (45)	1766 (1479)
expr	3370 (2264)	1427 (1181)	308 (299)	720 (709)	2588 (2319)

Table 8.12: **Predictions:** Predictions for ORFs of unknown function (classes 99,0,0,0 and 98,0,0,0). Numbers of predictions made are given with actual numbers of ORFs in brackets, as there may be more than one class predicted for each ORF. All rules produced were used.

	level				
datatype	1	2	3	4	all
seq	2240 (1646)	39 (39)	38 (38)	0 (0)	156 (147)
pheno	25 (25)	64 (64)	44 (44)	0 (0)	0 (0)
struc	114 (114)	109 (99)	0 (0)	0 (0)	29 (27)
hom	133 (82)	325 (301)	4 (4)	13 (13)	49 (48)
cellcycle	993 (961)	785 (748)	392 (392)	0 (0)	1910 (1544)
church	4 (4)	75 (49)	0 (0)	0 (0)	1333 (1079)
derisi	1164 (1144)	148 (129)	0 (0)	0 (0)	74 (59)
eisen	9 (9)	32 (24)	15 (15)	0 (0)	15 (13)
gasch1	918 (873)	737 (714)	35 (35)	21 (21)	1232 (1065)
gasch2	203 (201)	212 (194)	12 (12)	0 (0)	1732 (1522)
spo	174 (174)	116 (104)	0 (0)	0 (0)	221 (210)
expr	1133 (1066)	1416 (1175)	150 (149)	0 (0)	52 (42)

Table 8.13: **Predictions:** Predictions for ORFs of unknown function (classes 99,0,0,0 and 98,0,0,0) made by VALIDATED rulesets. Numbers of predictions made are given with actual numbers of ORFs in brackets, as there may be more than one class predicted for each ORF. Only rules which were statistically significant on the validation set are used. Significance was calculated by the hypergeometric distribution, with alpha=0.05 and Bonferroni correction.

8.5.4 Number of rules

The number of rules produced by each ruleset is generally small, often less than 10 for the validated rulesets (see Tables 8.14 and 8.15 for all rules and validated rules respectively). We then wanted to know: were these rules simply a complicated way of picking up deep homology relationships, or are they more general than homology? Could these results be obtained simply by clustering the results of sequence similarity searches? So we performed a PSI-BLAST search of yeast ORFs against themselves to find all homologous relationships between yeast ORFs. We then clustered the ORFs that fit each rule in turn, to see if we were simply picking up one homology cluster or unrelated ORFs. Table 8.16 shows how many rules in the validated rulesets were actually predicting more than one homology cluster, and Table 8.17 shows how many rules were predicting new homology clusters on the test data (i.e. the test data ORFs that matched the rule were not homologous to any of the training data ORFs that matched that rule). Most of the rules are predicting both more than one homology class and new homology classes, so our rules *are more general than possible using homology.*

	level				
datatype	1	2	3	4	all
seq	12	23	5	6	31
pheno	33	37	16	10	34
struc	10	9	6	2	11
hom	13	16	11	1	14
cellcycle	12	13	12	5	11
church	12	15	12	9	21
derisi	6	14	7	5	15
eisen	11	19	7	5	19
gasch1	12	23	12	5	17
gasch2	11	16	8	6	8
spo	12	13	8	3	13
expr	10	13	11	5	13

Table 8.14: Number of rules produced for individual datasets.

	level				
datatype	1	2	3	4	all
seq	10	4	2	0	13
pheno	4	4	1	0	2
struc	2	5	0	0	3
hom	9	12	3	1	7
cellcycle	6	8	3	0	6
church	3	3	0	0	8
derisi	2	6	0	0	5
eisen	3	11	3	0	7
gasch1	8	13	3	1	9
gasch2	4	7	1	0	7
spo	2	5	0	0	4
expr	7	10	3	0	7

Table 8.15: Number of VALIDATED rules produced for individual datasets. Only rules which were statistically significant on the validation set are used. Significance was calculated by the hypergeometric distribution, with alpha=0.05 and Bonferroni correction.

	level				
datatype	1	2	3	4	all
seq	7	3	1	0	7
pheno	4	4	1	0	2
struc	1	1	0	0	1
hom	5	7	1	0	5
cellcycle	6	8	2	0	6
church	3	3	0	0	8
derisi	2	6	0	0	5
eisen	3	11	3	0	7
gasch1	8	13	3	1	9
gasch2	4	7	1	0	6
spo	2	5	0	0	4
expr	7	10	3	0	7

Table 8.16: Number of VALIDATED rules predicting MORE THAN ONE HOMOLOGY CLASS. Only rules which were statistically significant on the validation set are used. Significance was calculated by the hypergeometric distribution, with alpha=0.05 and Bonferroni correction.

	level				
datatype	1	2	3	4	all
seq	7	3	0	0	7
pheno	4	4	1	0	2
struc	1	1	0	0	0
hom	2	5	1	0	4
cellcycle	6	8	1	0	6
church	3	3	0	0	8
derisi	2	6	0	0	5
eisen	3	10	3	0	7
gasch1	8	12	3	0	8
gasch2	4	6	0	0	6
spo	2	4	0	0	4
expr	7	10	3	0	7

Table 8.17: Number of VALIDATED rules predicting A NEW HOMOLOGY CLASS. A homology class is new if it is found only in the test data ORFs, and not in the training or validation data. Only rules which were statistically significant on the validation set are used. Significance was calculated by the hypergeometric distribution, with alpha=0.05 and Bonferroni correction.

8.6 Combinations of data

It is well known in machine learning that methods for voting classification algorithms can improve machine learning accuracy (Dietterich, 2000; Bauer & Kohavi, 1999). We wanted to try voting strategies and also direct combination of different types of data before learning, to see if results could be improved. In this section we report results on direct combination and in the following section we report our results of voting.

In the following experiments we used pairwise combination of datasets. The datasets were combined before C4.5 training. The datasets **seq**, **pheno**, **struc**, **hom** and **expr** were combined, making 10 possible pair combinations (**ceho**, **ceph**, **cese**, **cest**, **seho**, **seph**, **sest**, **stho**, **phho**, and **phst** - the names are constructed from the first two letters of the component datasets, except the expression set which uses the letters **ce**) . We also tried the combination of all 5 datasets (**all**).

8.6.1 Accuracy

The average accuracies of the validated rulesets in Table 8.19 range between 75% and 36% on level 1, dropping on the lower levels to 0% at level 4 where data is sparse. This is much the same as the accuracies we obtained on the individual data sets. Tables 8.20, 8.21 and 8.22 give class by class breakdowns for the higher levels, along with *a priori* class probabilities for comparison. Some classes are obviously predicted better than others, and some types of data predict certain classes better than others.

Class 5,0,0,0 - "protein synthesis" (and in particular 5,1,0,0 - "ribosome biogenesis") is again predicted very well by most datasets. Class 40,3,0,0 - "cytoplasm" likewise, since many ORFs which belong to 5,1,0,0 also belong to 40,3,0,0. 8,4,0,0 - "mitochondrial transport" is predicted very strongly by several datasets, mostly those that contain homology data. **ceph** predicts 40,16,0,0 - "mitochondrion". Many datasets predict 67,0,0,0 - "transport facilitation" well. **stho** predicts 29,0,0,0 - "transposable elements, viral and plasmid proteins" well. Class 1,0,0,0 - "metabolism" is predicted much better by the compound datasets than by the individual datasets.

The accuracy of the compound dataset results varies from class to class and from dataset to dataset, but generally seems to be about the same as the accuracy of the individual datasets.

	level				
datatype	1	2	3	4	all
ceho	67	59	43	0	62
ceph	50	39	27	45	53
cese	53	41	27	15	48
cest	54	47	45	32	58
seho	54	46	51	25	57
seph	55	50	50	25	65
sest	54	45	61	20	62
stho	56	47	39	14	63
phho	64	41	33	34	61
phst	57	29	37	20	54
all	59	59	39	28	54

Table 8.18: **Accuracy**: Average accuracy (percentages) on the test data of each ruleset produced by the compound datasets. All generated rules are included. Level "all" indicates the results of the hierarchical version of C4.5, which had classes from all levels in its training data.

	level				
datatype	1	2	3	4	all
ceho	74	65	66	0	87
ceph	36	39	34	0	68
cese	53	41	50	0	48
cest	45	47	41	0	65
seho	56	48	61	0	77
seph	45	78	42	0	71
sest	43	45	57	0	83
stho	47	47	47	0	68
phho	75	42	49	40	58
phst	47	30	0	0	36
all	52	60	38	0	56

Table 8.19: **Accuracy**: Average accuracy (percentages) on the test data of each VALIDATED ruleset produced by the compound datasets. Only rules which were statistically significant on the validation set are included. Level "all" indicates the results of the hierarchical version of C4.5, which had classes from all levels in its training data. Significance was calculated by the hypergeometric distribution, with alpha=0.05 and Bonferroni correction.

class	prior	ceho	ceph	cese	cest	seho	seph	sest	stho	phho	phst	all
1,0,0,0	27	80	47			78	48	36	49	76		47
2,0,0,0	6		42	49						20		
3,0,0,0	16		27	59			23				26	
4,0,0,0	20		33	31		35			41	72		
5,0,0,0	9	84	93	80	83	76	75	74		83		75
6,0,0,0	15			38						100		
8,0,0,0	12				30	45		27				
29,0,0,0	3	53		85		83	85	77	100	48		100
40,0,0,0	57	76		62		86						
67,0,0,0	8	64		55		74	80	90		81	66	

Table 8.20: Class by class accuracies for compound datasets, level 1, VALIDATED rules only.

class	prior	ceho	ceph	cese	cest	seho	seph	sest	stho	phho	phst	all
1,1,0,0	5	46										
1,5,0,0	11			38	32							
2,13,0,0	2	64	50		33							
4,1,0,0	3				31							
4,5,0,0	15		30									
5,1,0,0	5	69	86	90	73	78	82	65	50	40		56
5,10,0,0	1	100								18		46
6,13,0,0	4		42									
8,4,0,0	2	100				100		36	100	100	100	85
11,7,0,0	3							55			50	57
40,2,0,0	4			60			60		40			29
40,3,0,0	15	81	77	77	69	79	79	82	39	44		74
40,7,0,0	4		50									
40,10,0,0	20		28	30		34		29			26	
40,16,0,0	9		83	55	36							
67,28,0,0	1	18				18		55	60	18	50	57
67,50,0,0	2					33			67			

Table 8.21: Class by class accuracies for compound datasets, level 2, VALIDATED rules only.

class	prior	ceho	ceph	cese	cest	seho	seph	sest	stho	phho	phst	all
1,0,0,0	27						48		76		36	
3,0,0,0	16											42
4,0,0,0	20			30								
5,0,0,0	9	96	85	97	92	92	67	93		57		72
5,1,0,0	5	89	77	97	46	64	79	71	50	87		65
29,0,0,0	3			55		100	75	78	45	45		25
40,0,0,0	57		65	57						72		
40,3,0,0	14	88	70	83	72	81	86	92		43		79
40,10,0,0	20			27								
67,0,0,0	9	82		88			85					

Table 8.22: Class by class accuracies for compound datasets, all levels (hierarchical learning), VALIDATED rules only.

8.6.2 Coverage

Coverage varies less widely on compound data sets than on the individual datasets (see Tables 8.23 and 8.24 for all rules and validated rules respectively). **cese** has the greatest coverage, which is to be expected as the expression and **seq** datasets had the greatest coverage before. This makes some prediction for 79% of test data ORFs (1061 ORFs). Coverage is again poor at level 4 (usually 0).

datatype	level				
	1	2	3	4	all
ceho	71.17 (948)	11.01 (143)	10.80 (96)	0.54 (2)	81.68 (1088)
ceph	95.75 (1240)	36.50 (472)	16.61 (147)	5.43 (20)	86.80 (1124)
cese	79.21 (1063)	46.60 (610)	14.14 (126)	3.49 (13)	85.02 (1141)
cest	71.56 (946)	19.94 (261)	5.72 (51)	23.86 (89)	74.51 (985)
seho	69.45 (932)	26.20 (343)	9.43 (84)	15.55 (58)	57.90 (777)
seph	78.39 (1052)	12.68 (166)	2.58 (23)	8.85 (33)	42.62 (572)
sest	68.70 (922)	23.45 (307)	2.02 (18)	5.36 (20)	69.23 (929)
stho	60.63 (813)	14.14 (185)	7.30 (65)	1.61 (6)	54.88 (736)
phho	61.70 (815)	11.72 (151)	17.67 (156)	43.24 (160)	44.28 (585)
phst	78.36 (1032)	20.17 (263)	2.03 (18)	1.35 (5)	49.73 (655)
all	65.20 (875)	11.23 (147)	10.89 (97)	15.01 (56)	74.66 (1002)

Table 8.23: **Coverage**: Test set coverage of each ruleset produced by the compound datasets. Figures are given in percentages with actual numbers of ORFs in brackets. All generated rules are included. Level "all" indicates the results of the hierarchical version of C4.5, which had classes from all levels in its training data.

datatype	level				
	1	2	3	4	all
ceho	31.91 (425)	9.24 (120)	4.27 (38)	0.00 (0)	6.61 (88)
ceph	41.70 (540)	35.81 (463)	6.55 (58)	0.00 (0)	40.69 (527)
cese	79.06 (1061)	45.23 (592)	1.57 (14)	1.34 (5)	75.48 (1013)
cest	10.44 (138)	14.06 (184)	4.38 (39)	0.00 (0)	4.77 (63)
seho	36.89 (495)	24.68 (323)	3.14 (28)	0.00 (0)	5.14 (69)
seph	27.42 (368)	5.12 (67)	1.35 (12)	0.00 (0)	13.26 (178)
sest	23.92 (321)	22.15 (290)	0.79 (7)	0.00 (0)	5.44 (73)
stho	18.05 (242)	13.53 (177)	3.82 (34)	0.00 (0)	5.44 (73)
phho	18.70 (247)	9.86 (127)	4.19 (37)	1.35 (5)	16.81 (222)
phst	6.83 (90)	15.34 (200)	0.00 (0)	0.00 (0)	12.98 (171)
all	26.38 (354)	10.62 (139)	2.36 (21)	0.00 (0)	14.75 (198)

Table 8.24: **Coverage**: Test set coverage of each VALIDATED ruleset produced by the compound datasets. Figures are given in percentages with actual numbers of ORFs in brackets. Only rules which were statistically significant on the validation set are included. Level "all" indicates the results of the hierarchical version of C4.5, which had classes from all levels in its training data. Significance was calculated by the hypergeometric distribution, with alpha=0.05 and Bonferroni correction.

8.6.3 Predictions

The numbers of predictions can be found in Tables 8.25 and 8.26 (for all rules and validated rules respectively). **cese** and **ceph** make by far the most validated predictions (1,555 and 1,187 ORFs respectively). **cese** is expected since it has the greatest test set coverage, but **ceph** is something of a surprise. When the ruleset is examined, it can be seen that it uses the expression data only, and the phenotype data is not used in any rules. Since the phenotype data is very sparse, this is to be expected - it will not in general have a greater discrimination than expression data if fewer examples are covered.

	level				
datatype	1	2	3	4	all
ceho	1238 (1213)	83 (64)	82 (80)	10 (10)	1980 (1943)
ceph	4136 (2266)	1316 (1090)	451 (394)	95 (93)	2923 (1987)
cese	2250 (1593)	800 (775)	605 (596)	145 (145)	3216 (1833)
cest	1766 (1591)	349 (294)	36 (36)	196 (185)	2061 (1981)
seho	1256 (944)	250 (245)	34 (34)	204 (202)	584 (471)
seph	1994 (1636)	226 (222)	34 (34)	334 (314)	590 (566)
sest	1891 (1605)	307 (305)	19 (19)	35 (35)	1526 (1513)
stho	1278 (1200)	64 (46)	28 (28)	24 (24)	958 (954)
phho	467 (429)	68 (38)	105 (105)	329 (328)	1083 (1062)
phst	1900 (1812)	450 (414)	13 (13)	27 (27)	1390 (1269)
all	1287 (1113)	58 (38)	59 (58)	373 (372)	910 (719)

Table 8.25: **Predictions:** Predictions for ORFs of unknown function (classes 99,0,0,0 and 98,0,0,0). Numbers of predictions made are given with actual numbers of ORFs in brackets, as there may be more than one class predicted for each ORF. All rules produced were used.

	level				
datatype	1	2	3	4	all
ceho	100 (98)	72 (57)	21 (21)	0 (0)	40 (27)
ceph	1898 (1187)	1302 (1076)	133 (133)	0 (0)	580 (561)
cese	2207 (1555)	769 (769)	29 (29)	98 (98)	2679 (1585)
cest	126 (122)	247 (198)	31 (31)	0 (0)	80 (70)
seho	341 (314)	227 (227)	9 (9)	0 (0)	16 (11)
seph	564 (556)	2 (2)	33 (33)	0 (0)	151 (147)
sest	546 (533)	294 (292)	6 (6)	0 (0)	17 (17)
stho	116 (113)	58 (46)	6 (6)	0 (0)	38 (38)
phho	71 (71)	62 (35)	14 (14)	18 (18)	119 (105)
phst	112 (112)	318 (307)	0 (0)	0 (0)	281 (281)
all	401 (401)	55 (37)	3 (3)	0 (0)	94 (71)

Table 8.26: **Predictions:** Predictions for ORFs of unknown function (classes 99,0,0,0 and 98,0,0,0) made by VALIDATED rulesets. Numbers of predictions made are given with actual numbers of ORFs in brackets, as there may be more than one class predicted for each ORF. Only rules which were statistically significant on the validation set are used. Significance was calculated by the hypergeometric distribution, with alpha=0.05 and Bonferroni correction.

8.6.4 Number of rules

Again, the number of rules produced by each ruleset is generally small, often less than 10 for the validated rulesets (see Tables 8.27 and 8.28 for all rules and validated rules respectively). Tables 8.29 and 8.30 show how many rules predicted more than one homology class and new homology classes, as described in Section 8.5.4.

In comparison with the overall number of rules, fewer rules predicted more than one homology class or new homology classes, but even so, this is still more than half the rules.

	level				
datatype	1	2	3	4	all
ceho	13	11	10	1	7
ceph	11	11	11	4	12
cese	14	17	11	2	24
cest	10	15	6	4	8
seho	13	12	8	3	8
seph	15	13	5	6	31
sest	7	8	4	2	7
stho	5	10	8	2	5
phho	10	9	11	4	7
phst	7	12	5	2	5
all	4	12	11	3	9

Table 8.27: Number of rules produced for compound datasets.

	level				
datatype	1	2	3	4	all
ceho	9	8	3	0	5
ceph	8	9	2	0	8
cese	12	7	2	1	11
cest	3	7	1	0	3
seho	10	7	2	0	4
seph	11	3	2	0	11
sest	6	6	1	0	5
stho	4	6	4	0	3
phho	9	5	5	1	6
phst	2	4	0	0	1
all	3	8	3	0	5

Table 8.28: Number of VALIDATED rules produced for compound datasets. Only rules which were statistically significant on the validation set are used. Significance was calculated by the hypergeometric distribution, with alpha=0.05 and Bonferroni correction.

	level				
datatype	1	2	3	4	all
ceho	6	5	1	0	4
ceph	8	9	2	0	8
cese	7	6	2	1	6
cest	3	7	1	0	3
seho	6	3	1	0	3
seph	3	2	1	0	5
sest	4	4	0	0	3
stho	3	2	1	0	2
phho	6	2	3	0	4
phst	2	1	0	0	1
all	2	3	1	0	3

Table 8.29: Number of VALIDATED rules predicting MORE THAN ONE HOMOLOGY CLASS. Only rules which were statistically significant on the validation set are used. Significance was calculated by the hypergeometric distribution, with alpha=0.05 and Bonferroni correction.

	level				
datatype	1	2	3	4	all
ceho	5	5	0	0	4
ceph	8	9	2	0	8
cese	6	5	1	0	6
cest	2	7	1	0	3
seho	4	3	1	0	3
seph	3	2	0	0	5
sest	2	3	0	0	3
stho	3	2	0	0	2
phho	3	2	2	0	4
phst	2	1	0	0	1
all	2	3	0	0	3

Table 8.30: Number of VALIDATED rules predicting A NEW HOMOLOGY CLASS. A homology class is new if it is found only in the test data ORFs, and not in the training or validation data. Only rules which were statistically significant on the validation set are used. Significance was calculated by the hypergeometric distribution, with alpha=0.05 and Bonferroni correction.

8.7 Voting strategies

Direct combination of the data before learning is one method of making use of multiple data sources. Another is to learn separate classifiers for each of the data sources, and then combine their results in some way (Ali & Pazzani, 1996). There are various strategies for combining the results and here we investigate several voting strategies.

8.7.1 Strategies

We have several rulesets, one produced from each of the individual datasets, one from each pair of individual datasets, and one produced from the combination of all data. To allow these rulesets to vote for the class of an ORF we need to consider several problems.

- First, each ruleset may have more than one rule that predicts a class for an ORF. A ruleset may contain several rules that all predict the same class for an ORF (which could be seen as duplicating a prediction or reinforcing a prediction). Or it may predict several different classes for an ORF. All may be valid, since an ORF may have more than one class.
- Second, each rule comes with a confidence value - the accuracy that the rule had on the validation set. This gives us a measure of how general this rule will be when applied to unseen data, and we may want to use this either to weight the vote, or to decide which rules have the right to vote.

So we have many different voting mechanisms that could be applied. Here we list just a few:

- **non-weighted best rule only**
  The best rule only from each ruleset has a single vote
- **weighted best rule only**
  The best rule only from each ruleset has a weighted vote
- **non-weighted reinforcement**
  All rules from all rulesets have a single vote (duplicate predictions reinforce)
- **non-weighted no reinforcement**
  All rules from all rulesets have a single vote, but duplicate predictions within a ruleset have no additional effect
- **weighted reinforcement**
  All rules from all rulesets have a weighted vote (duplicate predictions reinforce)

- **weighted no reinforcement**
  All rules from all rulesets have a weighted vote, but duplicate predictions within a ruleset have no additional effect (only the best is chosen).

Should two votes at 50% confidence each be equivalent to one vote at 100% confidence? If we allow a weighted sum then they would be, however if we allow non-weighted voting they would be worth twice as much. Bayesian combination would have been a possibility if we had both weights associated with the rules *and* weights associated with the rulesets, but we only have the former. Average validation set accuracy could be used as the weight of a ruleset, but this would be a fairly meaningless value. For example, a ruleset with a low average accuracy could have that low accuracy because of just one rule, but could still be the best predictor of other classes.

Since we allow ORFs to have more than one function we also have the issue of determining the result of the voting - do we take all possible candidates as the predictions or only the best? Do we use all predictions which reach a certain voting threshold, or do we use all, regardless of threshold? This is the standard problem of trading off coverage against accuracy. We can make more predictions at lower accuracy or fewer predictions at higher accuracy.

Tables 8.31, 8.32, 8.33, 8.34 and 8.35 show the results of several of the above-mentioned voting strategies on the individual datasets at level 2 only for comparison. We have **weighted reinforcement** (Table 8.31), **weighted no reinforcement** (Table 8.33), **non-weighted reinforcement** (Table 8.33) and **non-weighted no reinforcement** (Table 8.34). Table 8.35 shows **weighted best rule only voting**. The tables show class-by-class accuracies, overall average accuracy, and overall coverage. We can see from these tables that reinforcement or non-reinforcement makes little difference. However weighting by validation set accuracy does help. This allows the confidence of a rule to be taken into account. Using all rules improves slightly on using the best rule only. Therefore, for all future voting by rulesets, the voting strategy used will be weighted reinforcement, all applicable rules.

	sum of confidences (%) (≥)			
class	50	100	150	200
1/1/0/0	60			
1/3/0/0	46			
2/13/0/0	58			
4/5/0/0	67			
5/1/0/0	77	90	95	100
6/13/0/0	42			
8/4/0/0	92	100	100	
30/1/0/0	83	83	100	100
40/2/0/0	60			
40/3/0/0	63	91	91	100
40/10/0/0	28	0		
40/16/0/0	63	100		
67/28/0/0	100			
67/50/0/0	100	100		
overall average	55	85	94	100
coverage	326 (419)	94 (123)	49 (77)	27 (36)

Table 8.31: Level 2, individual datasets. Weighted reinforcement voting, all applicable rules. Accuracy is shown in percentages on a class by class basis, and an overall average accuracy at the bottom. Coverage shows number of test set ORFs predicted, with number of predictions made in brackets. There were 1310 test set ORFS in total.

	sum of confidences (%) (≥)		
class	50	100	150
1/1/0/0	60		
1/3/0/0	46		
2/13/0/0	58		
4/5/0/0	67		
5/1/0/0	77	90	97
6/13/0/0	42		
8/4/0/0	92	100	100
30/1/0/0	83	83	
40/2/0/0	60		
40/3/0/0	63	91	91
40/10/0/0	28	0	
40/16/0/0	50	100	
67/28/0/0	100		
67/50/0/0	100	100	
overall average	55	84	94
coverage	322 (407)	93 (122)	43 (70)

Table 8.32: Level 2, individual datasets. Weighted no-reinforcement voting, all applicable rules. Accuracy is shown in percentages on a class by class basis, and an overall average accuracy at the bottom. Coverage shows number of test set ORFs predicted, with number of predictions made in brackets. There were 1310 test set ORFS in total.

	number of votes (≥)		
class	1	2	3
1/1/0/0	60		
1/3/0/0	46		
2/13/0/0	58		
4/5/0/0	33	67	
5/1/0/0	77	94	100
5/10/0/0	83		
6/13/0/0	42		
8/4/0/0	92	100	
8/19/0/0	50		
11/7/0/0	50		
30/1/0/0	83	100	
40/2/0/0	28	63	
40/3/0/0	63	91	100
40/10/0/0	30	29	0
40/16/0/0	27	63	100
67/10/0/0	31		
67/28/0/0	25	100	
67/50/0/0	100		
overall average	37	57	81
coverage	909 (1353)	201 (233)	33 (42)

Table 8.33: Level 2, individual datasets. Non-weighted reinforcement voting, all applicable rules. Accuracy is shown in percentages on a class by class basis, and an overall average accuracy at the bottom. Coverage shows number of test set ORFs predicted, with number of predictions made in brackets. There were 1310 test set ORFS in total.

	number of votes (≥)		
class	1	2	3
1/1/0/0	60		
1/3/0/0	46		
2/13/0/0	58		
4/5/0/0	33	67	
5/1/0/0	77	97	100
5/10/0/0	83		
6/13/0/0	42		
8/4/0/0	92	100	
8/19/0/0	50		
11/7/0/0	50		
30/1/0/0	83		
40/2/0/0	28	63	
40/3/0/0	63	91	100
40/10/0/0	30	29	0
40/16/0/0	27	50	100
67/10/0/0	31		
67/28/0/0	25	100	
67/50/0/0	100		
overall average	37	55	74
coverage	909 (1353)	184 (214)	22 (31)

Table 8.34: Level 2, individual datasets. Non-weighted no-reinforcement voting, all applicable rules. Accuracy is shown in percentages on a class by class basis, and an overall average accuracy at the bottom. Coverage shows number of test set ORFs predicted, with number of predictions made in brackets. There were 1310 test set ORFS in total.

	sum of confidences (%) (≥)			
class	50	100	150	200
1/1/0/0	60			
1/3/0/0	46			
2/13/0/0	20			
4/5/0/0	75			
5/1/0/0	81	92	96	100
6/13/0/0	45			
8/4/0/0	92	100	100	
30/1/0/0	83	83	100	100
40/2/0/0	56			
40/3/0/0	66	60	60	
40/10/0/0	29	0.00		
40/16/0/0	66	100		
67/50/0/0	100	100		
overall average	56	83	93	100
coverage	314 (365)	89 (92)	40 (40)	13 (13)

Table 8.35: Level 2, individual datasets. Weighted reinforcement voting, best rule only. Accuracy is shown in percentages on a class by class basis, and an overall average accuracy at the bottom. Coverage shows number of test set ORFs predicted, with number of predictions made in brackets. There were 1310 test set ORFS in total.

8.7.2 Accuracy and coverage

Voting from the different datasets can certainly be used to increase the accuracy and coverage of the results. Voting from each of the individual datasets (**seq**, **pheno**, **struc**, **hom** and **expr**) gives an average accuracy of at least 61% on level 1 (see Table 8.36) and at least 55% on level 2 (see Table 8.31) of the test set. These can be tuned further to give higher accuracy at the cost of lower coverage. By comparison, direct combination of all 5 datasets before learning gives an accuracy of 52% at level 1 and 60% at level 2, with only 1/3 of the coverage in each case. At a similar level of coverage, the voting would give accuracies of 80% on level 1 and 85% on level 2. We also find that a wider variety of classes can be predicted by the voting method than by direct combination before learning.

The only drawback of the voting method is that the results become more difficult to interpret. If an ORF is predicted to belong to a particular class we can see which rules have voted for this class, but then we have to take into account all these rules when trying to understand the biological explanation for the prediction.

Tables 8.36 and 8.31 show the results for levels 1 and 2 of voting from the **seq**, **pheno**, **struc**, **hom** and **expr** datasets. Tables 8.37 and 8.38 show the results of voting from all the individual expression data sets, and Tables 8.39 and 8.40 show the results of voting from all the paired datasets.

The results from the expression data voting are not as accurate as the results from the other types of data. But this data is known to be noisy, and it is also probably more self-similar than the other types of data, so voting will not be able to gain so much. However, accuracy and coverage are still improved from direct combination of expression data sets (see the **expr** lines in Tables 8.6 and 8.11 for comparison with Tables 8.37 and 8.38).

	sum of confidences (%) (≥)			
class	50	100	150	200
1/0/0/0	52	70	91	93
29/0/0/0	68	82	67	100
3/0/0/0	75			
30/0/0/0	75	100	100	
4/0/0/0	41			
40/0/0/0	63	83	80	100
5/0/0/0	76	93	93	100
6/0/0/0	100	100		
67/0/0/0	75	77		
overall average	61	80	91	97
coverage	1013 (1400)	250 (280)	85 (86)	35 (35)

Table 8.36: Level 1, **individual** datasets. Weighted reinforcement voting, all applicable rules. Accuracy is shown in percentages on a class by class basis, and an overall average accuracy at the bottom. Coverage shows number of test set ORFs predicted, with number of predictions made in brackets. There were 1343 test set ORFS in total.

	sum of confidences (%) (≥)		
class	50	100	150
1/0/0/0	53		
2/0/0/0	42	60	
3/0/0/0	38	50	
5/0/0/0	44	89	95
6/0/0/0	35		
40/0/0/0	63	66	64
overall average	58	69	80
coverage	1030 (1335)	487 (515)	116 (119)

Table 8.37: Level 1, **individual expression** datasets. Weighted reinforcement voting, all applicable rules. Accuracy is shown in percentages on a class by class basis, and an overall average accuracy at the bottom. Coverage shows number of test set ORFs predicted, with number of predictions made in brackets. There were 1343 test set ORFS in total.

	sum of confidences (%) (≥)			
class	50	100	150	200
1/2/0/0	25			
2/13/0/0	36	67		
3/1/0/0	50			
3/3/0/0	50			
4/1/0/0	50			
5/1/0/0	57	72	81	90
6/13/0/0	55	100	100	
40/3/0/0	41	62	77	86
40/7/0/0	58	100		
40/10/0/0	38	39		
40/16/0/0	48	82	100	100
overall average	44	64	79	88
coverage	537 (669)	179 (252)	101 (157)	75 (124)

Table 8.38: Level 2, **individual expression** datasets. Weighted reinforcement voting, all applicable rules. Accuracy is shown in percentages on a class by class basis, and an overall average accuracy at the bottom. Coverage shows number of test set ORFs predicted, with number of predictions made in brackets. There were 1310 test set ORFS in total.

	sum of confidences (%) (≥)					
class	50	100	150	200	250	300
1/0/0/0	56	73	77	84	85	79
2/0/0/0	53					
3/0/0/0	31	75	100			
4/0/0/0	39	51	73	67		
5/0/0/0	69	77	77	80	80	89
6/0/0/0	44	100	100			
8/0/0/0	31	67				
29/0/0/0	50	69	82	88	95	100
40/0/0/0	63	83	90	94	90	100
67/0/0/0	66	76	78	86	87	89
overall average	55	75	80	85	85	90
coverage	1343 (1978)	490 (591)	277 (348)	212 (259)	179 (196)	126 (139)

Table 8.39: Level 1, **compound (paired)** datasets. Weighted reinforcement voting, all applicable rules. Accuracy is shown in percentages on a class by class basis, and an overall average accuracy at the bottom. Coverage shows number of test set ORFs predicted, with number of predictions made in brackets. There were 1343 test set ORFS in total.

	sum of confidences (%) (≥)			
class	50	100	150	200
1/5/0/0	58			
2/13/0/0	53	64	100	
4/1/0/0	50			
5/1/0/0	48	63	77	82
5/10/0/0	36	100		
6/13/0/0	41.67			
8/4/0/0	100	100	100	100
11/7/0/0	67			
67/28/0/0	24	60	100	
67/50/0/0	67			
40/2/0/0	57	60	0	
40/3/0/0	50	53	75	76
40/10/0/0	33	42	20	
40/16/0/0	89	92	100	
overall average	43	58	77	81
coverage	611 (778)	220 (318)	109 (172)	88 (140)

Table 8.40: Level 2, **compound (paired)** datasets. Weighted reinforcement voting, all applicable rules. Accuracy is shown in percentages on a class by class basis, and an overall average accuracy at the bottom. Coverage shows number of test set ORFs predicted, with number of predictions made in brackets. There were 1310 test set ORFS in total.

The compound datasets voting (using all 10 pairwise combinations of **seq**, **pheno**, **struc**, **hom** and **expr**) performs slightly better than the individual dataset voting at level 1 but slightly worse at level 2. The relationship between direct combination before learning, individual dataset voting and compound dataset voting can be seen in Figures 8.5 and 8.6, for levels 1 and 2 respectively.

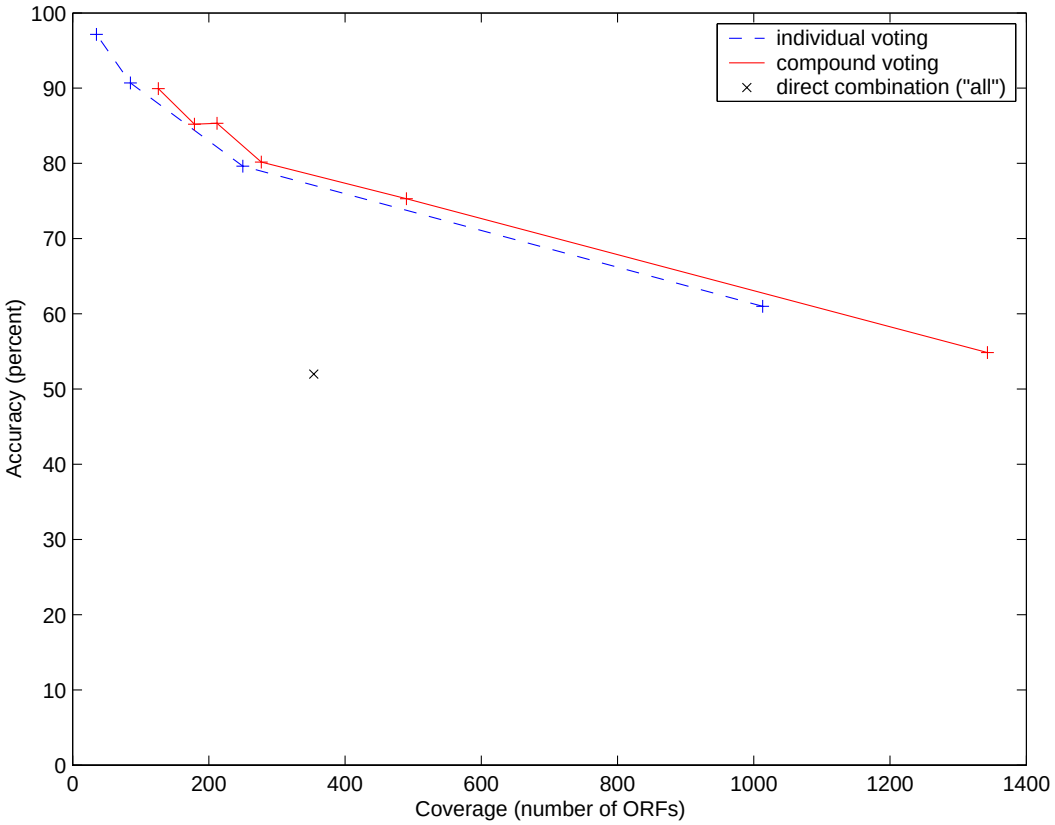


Figure 8.5: Coverage and accuracy at level 1. The single cross is from the direct combination before learning of **seq**, **pheno**, **struc**, **hom** and **expr**. The simple voting is voting from **seq**, **pheno**, **struc**, **hom** and **expr**. The compound voting is from all 10 of the the pairwise combinations of these datasets.

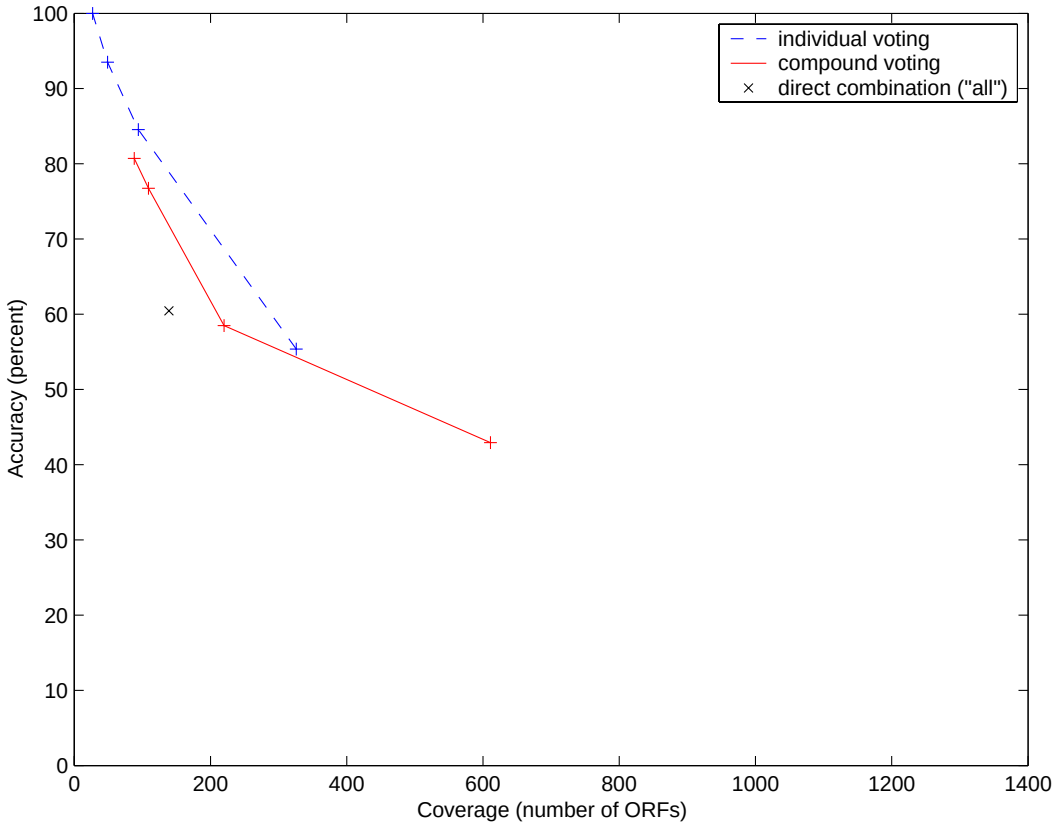


Figure 8.6: Coverage and accuracy at level 2. The single cross is from the direct combination before learning of **seq**, **pheno**, **struc**, **hom** and **expr**. The simple voting is voting from **seq**, **pheno**, **struc**, **hom** and **expr**. The compound voting is from all 10 of the the pairwise combinations of these datasets.

8.8 Biological understanding

The rules can be used to understand more about the biology behind the predictions. Here we demonstrate that the rules can be shown to be consistent with known biology.

```
ss(_1,_2,c),ss(_1,_3,c),ss(_1,_4,b),nss(_2,_5,b),nss(_5,_6,c) = 0
ss(_1,_2,c),ss(_1,_3,a),alpha_len(_3,b10_14),coil_len(_2,b3_4),nss(_2,_3,a) = 1
ss(_1,_2,c),ss(_1,_3,a),alpha_len(_3,b10_14),coil_len(_2,b1_3),nss(_2,_3,a) = 1
ss(_1,_2,c),ss(_1,_3,a),alpha_len(_3,b3_6),coil_len(_2,b3_4),nss(_2,_3,a) = 1
ss(_1,_2,c),ss(_1,_3,a),alpha_len(_3,b1_3),coil_len(_2,b6_10),nss(_2,_3,a) = 0
->  class 8/4/0/0 "mitochondrial transport"
```

Figure 8.7: Rule 76, level 2, **struc** data

Figure 8.7 shows a rule from level 2 of the structure data. This rule is 80% accurate on the test set.

This rule means the following:

- no: coil followed by beta followed by coil (c-b-c).
- yes: coil (of length 3) followed by alpha ($10 \leq$ length < 14).
- yes: coil (of length either 1 or 2) followed by alpha ($10 \leq$ length < 14).
- yes: coil (of length 3) followed by alpha ($3 \leq$ length < 6).
- no: coil ($6 \leq$ length < 10) followed by alpha (of length either 1 or 2).

So there are many short coils followed by longish alphas and this happens at least 3 times. There is no coil-beta-coil and there are no longish coils followed by short alphas.

In fact this rule predicts many of the MCF (Mitochondrial Carrier Family). These are known to have six transmembrane a-helical spanning regions (Kuan & Saier Jr., 1993). Kuan and Saier produced a multiple alignment of members of the MCF and analysed hydropathy plots. They observed that "These analyses revealed that the six transmembrane spanners exhibited varying degrees of sequence conservation and hydrophilicity. These spanners, and immediately adjacent hydrophilic loop regions, were more highly conserved than other regions of these proteins."

Figure 8.8 shows the 6 alpha helices spanning the membrane.

The alpha helices in these proteins are known to be long, in the order of 20-30 amino acids (Senes *et al.*, 2000). So we were curious to understand why this rule selects alpha stretches of only 10 to 14 amino acids. Using Clustal W

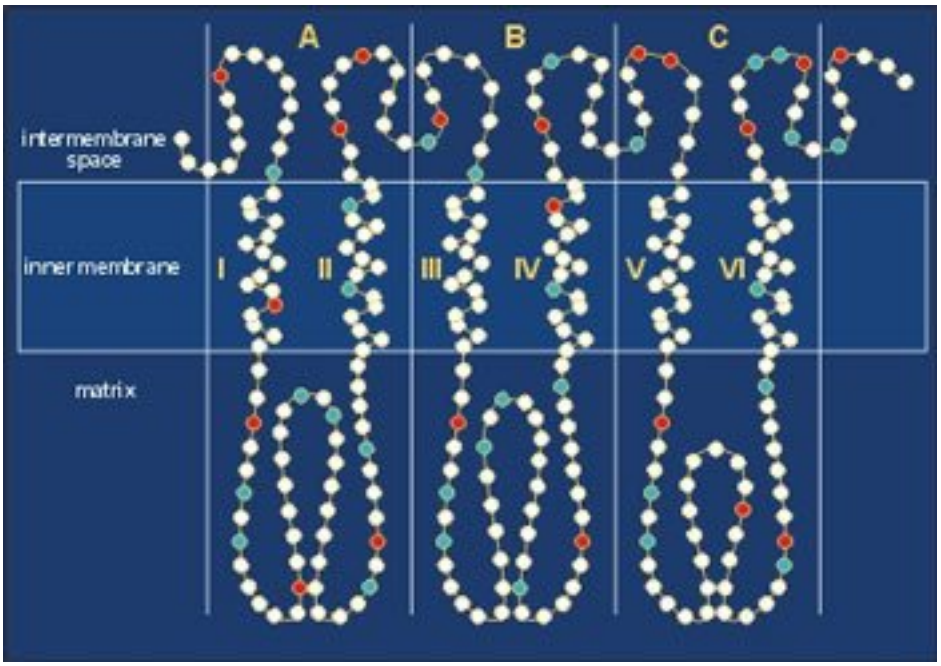


Figure 8.8: Topology model showing the six transmembrane helices of the mitochondrial transporters. Image from http://www.mrc-dunn.cam.ac.uk/research/transporters.html, by permission of Dr E. Kunji.

(Higgins *et al.*, 1994) for a multiple alignment of the sequences of all the ORFs covered by this rule showed a few consensus positions, but nothing immediately obvious. Overlaying the secondary structure predictions given by PROF onto the alignment showed a striking pattern. Each long alpha helix was broken in the middle by one or two short coils of 2-3 amino acids in length. All these short coils aligned perfectly and appeared at glycines and prolines in the sequences. Glycine is the smallest amino acid and may disrupt the helix. Proline is also known to cause kinks in helices, since it has an imino rather than an amino group. The rule detects these "kinks" in the helices.

The multiple sequence alignment of all ORFs that fit this rule except for the two errors of commission and the prediction YMR129W can be seen in Appendix C. The conserved glycines and prolines at the locations of the short helix-breaking coils are marked. Helices 1, 3 and 5 have a conserved proline, whereas helices 2, 4 and 6 have a conserved GxxxxxxG motif. This motif is known to be associated with transporter/channel like membrane proteins (Liu *et al.*, 2002).

Errors of commission of this rule are:

```
YMR288W (HSH155) ‘‘component of a multiprotein splicing factor’’
YHR190W (ERG9)   ‘‘lipid, fatty-acid and isoprenoid biosynthesis,
```

```
endoplasmic reticulum,
farnesyl-diphosphate farnesyltransferase''
```

These are quite certainly not members of the mitochondrial transport class. The rule is correct for the MCF members, but only recognises 3 parts of the alpha-helices, not all 6, and it is possible that there are other unrelated proteins that share this much of the structure.

This rule predicts two ORFs that are currently listed as being "unclassified proteins", YPR128C and YMR192W. One of these predictions has been shown to be correct. YPR128C has recently been shown to be a member of the mitochondrial carrier family (van Roermund *et al.*, 2001), although MIPS still does not make this classification. YMR192W, on the other hand, does not align well with the other sequences, either by primary or secondary structure, and we doubt that this prediction is correct.

8.9 Predictions

Many predictions have been made for ORFs whose functions are currently unknown. Some function is predicted by our validated rule sets for 2411 (96%) of the ORFs of unknown function. Each prediction is made by a rule with an estimated accuracy, so biologists will have an idea of the confidence of the prediction, and the rule may give an intuition into the biological reasons for the prediction.

All predictions are available from http://www.genepredictions.org. This is a database of gene function predictions, hosted at Aberystwyth. The database is intended to hold predictions for any organism, and to act as a free repository that can be accessed by anyone wanting information about the possible function of genes that do not currently have annotations in the standard databases.

Chapter 9

Conclusions

9.1 Conclusions

This work has extended the methods used to predict gene function in *M. tuberculosis* and *E. coli* to the larger and more data-rich genome of *S. cerevisiae*. Several challenges have been faced and solutions found.

Accurate and informative rules have been learnt for the functions of *S. cerevisiae* ORFs. These rules have made many predictions for ORFs of currently unknown function and we look forward to having these experimentally tested. The rules and predictions are now freely available on the web[1]. The following have also been achieved:

- Use of more of the full range of data available from biology. Data collection from publicly available databases around the World Wide Web, and validation, integrity testing and preprocessing of this data.
- Direct use of multiple class labels within C4.5.
- Use of hierarchical class information and hierarchically structured data.
- Development of relational data mining software for the distributed environment of a Beowulf cluster in order to address scaling problems due to the volume of data.
- Results from different data sources and algorithms were successfully combined and evaluated.

[1] Available at http://www.genepredictions.org

Data

Many different types of data about *S. cerevisiae* have been collected and used in this work. Collection of data was from public webservers around the world and this was sometimes made difficult by "point and click" interfaces. These interfaces generally allow querying by single genes, which means that extracting data for the whole genome can require special software to repeatedly query the website.

Properties of the amino acid sequence were easy to collect and proved to be a good predictor of protein function. Rules based on sequence properties had good coverage of the test set and made many predictions.

The phenotype dataset was a small and sparse dataset but was still useful in predicting several classes. Due to its small size it was less useful when combined with other datasets. When more phenotype data become available in future this should have more potential. This was to the best of our knowledge the first published work using phenotype data to predict functional class.

Data from microarray experiments on yeast were readily available on the internet but tended to be noisy, and not as reliable as expected. We expect the quality and standardisation of microarray experiments to improve in the near future.

Rules formed from predicted secondary structure were limited in their coverage. Some very accurate rules were produced, but perhaps the scope of the rules was limited by the range of patterns mined from the data before machine learning was applied. The patterns involved neighbouring structures, but not long range relationships, and overall structure distributions, but not more complex distributions.

Homology data provided a wide range of interesting and accurate rules that reflected the richness of the dataset. This was expected, and shows the strength of sequence similarity when inferring function.

All datasets collected and used in this work are made available on the web[2]. These will be useful as testbeds for future machine learning research.

Methods

A specific machine learning method has been developed which handles the problems provided by the phenotype data: many classes, multiple class labels per gene and the need to know accuracies of individual rules rather than the ruleset as a whole. This has involved developing multilabel extensions of C4.5 coupled with a rule selection and bootstrap sampling procedure to give a clearer picture of the rules.

Three different clustering methods were used to investigate the relationship between microarray data and known biology. Clusters produced from the data

[2]Available at http://www.aber.ac.uk/compsci/Research/bio/dss/

reflected some known biology, however, the majority of clusters have no obvious common annotations from the current ORF annotation schemes. We expect that microarray data presents new biological knowledge and that in time the annotation schemes will represent this. We conclude that unsupervised clustering is limited for this application, and recommend that deeper data analysis methods such as ILP need to be used in future.

The PolyFARM algorithm was developed in order to mine large volumes of relational data. This uses the distributed hardware of a Beowulf cluster to discover frequent first order associations in the data. Frequent associations were successfully mined from predicted secondary structure and homology databases.

The use of hierarchically structured data and classes was investigated. An extension to the C4.5 algorithm was developed which could learn from hierarchically structured classes and an extension to the PolyFARM algorithm was developed which could learn from hierarchically structured data.

9.2 Original contributions to knowledge

This thesis makes the following original contributions to knowledge:

Computer Science:

1. **An extension of C4.5 to multi-label learning[3].** The standard machine learning program C4.5 was extended to allow multiple class labels to be specified for each ORF. This is not common in the field of machine learning and would normally be handled either by producing many classifiers, one per class.

2. **A distributed first order association mining algorithm.** This is a version of the WARMR algorithm which is designed to allow processing to be distributed across a Beowulf cluster of computers without shared memory. This was necessary to process the volume of data associated with the yeast genome. PolyFARM is freely available for non-commercial use from http://www.aber.ac.uk/compsci/Research/bio/dss/polyfarm.

3. **An extension of C4.5 to hierarchical class learning.** C4.5 was modified again to be able to use a class hierarchy. There has been little prior work on hierarchical learning except in conjunction with document clustering and simple Bayesian methods. Our version of C4.5 can now deal with problems

[3]Suzuki *et al.* (2001) produced a similar idea to this, published in the same conference. Their method also works by altering the entropy of a decision tree algorithm, although in a different way, using their own decision tree algorithm rather than C4.5, and they do not handle post-pruning or formation of rules.

of both multiple and hierarchical class labels, which is a common real world problem.

4. **An extension of the distributed first order association mining algorithm which allows hierarchically structured attributes.** This was a method of directly implementing the hierarchically structured attributes rather than explicitly having to list all ancestors for each item. This was necessary to reduce the size of our yeast database by allowing the hierarchy to become background knowledge and to avoid duplication. This should provide faster access to the data than the Prolog-style alternative of allowing the hierarchy to be specified and searched by recursive relations.

Biology:

1. **An investigation into the use of clustering for analysing microarray data.** We provided a quantification of how well clusters are supported by the existing biological knowledge and showed that current algorithms produce clusters that do not, in general, agree well with annotated biology.

2. **Predictions for many of the ORFs of unknown function in yeast.** Accurate predictions have been made, and these will all be available shortly through the webserver http://www.genepredictions.org which is being established for the purpose of allowing access to our predictions and those of others in future. Our predictions are in the form of understandable rules and we hope that the information in these rules can be interpreted to enhance current understanding of gene function.

9.3 Areas for future work

There are many areas for improvements on the techniques used here:

- **The use of ILP on expression data should be investigated.** Relating the timepoints in the time series to each other should be a better way to make use of the information from microarray experiments, since straightforward decision tree learning ignores these relationships. Lee and De Raedt (2002) describe an ILP system and language (SeqLog) for describing and mining sequence data. They present some preliminary findings from mining a database of Unix command histories. Expression data would be an interesting dataset for their system.

- **Better discretisation methods.** There are many discretisation methods available, both supervised and unsupervised. Different methods of discretisation may help with accuracy of several of the learning techniques presented here.

- **Feature extraction/dimensionality reduction.** The Boolean attributes from the **hom** and **struc** datasets were so numerous that C4.5 had to be modified to accept them (the number of attributes was previously held in a short int which was too small). Having too many irrelevant attributes has an effect on classifier performance, and this is well studied in machine learning (Yang & Pedersen, 1997; Aha & Bankert, 1995). Feature selection is a field of research which attempts to select only the more useful features from a large dataset, in order to speed up processing time, to avoid noisy irrelevant attributes, and improve the learning accuracy.

- **Faster hierarchical C4.5.** While the version of C4.5 with hierarchical classes works, it runs very slowly, and should be re-engineered and optimised to make it usable.

- **Improvements to PolyFARM.** PolyFARM is in its infancy and there are many improvements which can be made. We would like to add support for more user-defined constraints, such as the ability to add new variables that are guaranteed not to unify with existing variables. We would like to add support for recursive definitions in the background knowledge and more of the functionality of Prolog. We would also like to add support for real-valued numbers, rather than having to discretise numerical data. And finally, the whole system should integrate with standard relational databases as well as Datalog databases, for convenience.

- **Confusion matrices and result reporting.** When reporting results of learning with multiple labels, many of the standard approaches such as confusion matrices have no meaning. Better ways to report details about the results of learning should be investigated.

- **Experimental confirmation of results.** We would like to have our yeast predictions tested and hopefully confirmed by wet biology.

Many more sources of data become available as time goes on. Data about the metabolome will be the next challenge, and data about protein-protein interactions, pathways and gene networks will also need advanced data mining technologies. Many other genomes are now available, and their data also awaits mining. There is much still to be learned from the human genome, the genomes of plants and animals and the many pathogenic organisms that cause disease.

9.4 Publications from the work in this thesis

- Clare, A. and King R.D. (2003) Data mining the yeast genome in a lazy functional language. In proceedings of *Practical Aspects of Declarative Lan-*

guages (PADL'03).

- Clare, A. and King R.D. (2002) How well do we understand the clusters found in microarray data? *In Silico Biol.* 2, 0046.
- Clare, A. and King R.D. (2002) Machine learning of functional class from phenotype data. *Bioinformatics* 18(1) 160-166.
- Clare, A. and King R.D. (2001) Knowledge Discovery in Multi-Label Phenotype Data. In proceedings of *ECML/PKDD 2001.*
- King, R.D., Karwath, A., Clare, A., and Dehaspe, L. (2001) The Utility of Different Representations of Protein Sequence for Predicting Functional Class. *Bioinformatics* 17(5) 445-454.
- King, R.D., Karwath, A., Clare, A., and Dehaspe, L. (2000) Accurate prediction of protein functional class in the *M. tuberculosis* and *E. coli* genomes using data mining. *Comparative and Functional Genomics* 17 283-293 (nb: volume 1 of CFG was volume 17 of Yeast).
- King, R.D., Karwath, A., Clare, A., and Dehaspe, L. (2000) Genome scale prediction of protein functional class from sequence using data mining. In: *The Sixth International Conference on Knowledge Discovery and Data Mining (KDD 2000).*

Appendix A

C4.5 changes: technical details

C4.5

The algorithm behind C4.5 is described in detail in the book "C4.5: programs for Machine Learning" (Quinlan, 1993) and the code is open source and freely available. C5 is commercial code, so it is not available to modify. The original code for C4.5 is available from http://www.cse.unsw.edu.au/~quinlan/. My modified C4.5 may be available on request (C4.5 contains a copyright notice stating that it is not to be distributed in any form without permission of J. R. Quinlan). This Appendix describes in more detail the modifications that were made, for someone familiar with C4.5.

Changing the data structures

The first changes were to allow more than one class to be handled in C4.5 data structures. The code that reads in the data file was changed to allow multiple classes separated by the '@' character, and a new file of simple accessor functions to the multiple classes was made.

Next the problem of handling multi-classes in trees was considered: First the types of the tree itself had to be changed to allow multi-classes in a leaf. This involved some memory allocation and management. No changes were needed in besttree.c, the top level code for growing trees and the windowing code, except minor types to make the windowing code compile. As we do not use the windowing code in this work, it was not altered. In building the tree (build.c): parts like "Find the most frequent class" in the FormTree function became harder. What we need is to find the most frequent *set* of classes. Statements such as "If all cases are of the same class or there are not enough cases to divide, the tree is a leaf" (also in the FormTree function) became harder to test. For instance, a data set where all objects are of class 'a' but some of them are also class 'b' would require further

division.

Instead of comparing the suggested class to the item's real class, we now had to ask if the suggested class is one of the correct classes for an item. This means that the error count (when the answer is no) does not quite reflect the whole story. In the case above, a rule predicting class 'a' would have an error of 0, but still not have taken account of the fact that some are also class 'b'. This also means that the errors can be 0 but still the difference between the number of cases and the number of cases with the best class set can be non-zero. So the error count had to be altered to reflect this difference.

Information theory and pruning

The gain/info calculations also needed to be altered. This was as described in section 4.4.

After the tree is grown, the tree is pruned. Again, in order to do this we had to find the most frequent set of classes in this branch (rather than the best single class) and how many items have this set of classes, in order to check if we would be better off pruning this branch and replacing by a leaf. So this was handled the same way as finding the most frequent set of classes while building the tree, but this time it was local to this branch of the tree.

Rules generation and evaluation

Rules do not need to predict more than one class, as we can just apply several rules to each data item, each predicting one class for that item. Previously, each path through the tree was a potential candidate for making a rule. In this modified version of C4.5 each path through the tree, and each class in the leaf of that path is a candidate for making a rule. Essentially, the algorithm does not change, but is just repeated for each class predicted by that leaf. Again we had to be careful with error counts and frequency tables, as they did not sum to the values they did before. As before, the conditions for each rule are pruned to make the rule accuracy better.

Secondly, some rules are later pruned, and the rest are sorted and a default class chosen. The rule pruning depends on their value, which in turn depends on true and false positives and true and false negatives, which needed to be counted slightly differently in the modified version.

The previous method for testing the set of rules was a trial and error procedure to discover which rules should be dropped. All ORFs were predicted by the best (most confident, lowest error) rule that matched them and the second best (second most confident) rule that matched (or default rule if there was no second best).

Then at the end, if there was a rule that was consistently worse than using the second best rule, it was dropped and the process repeated with the new ruleset. This continued until no more rules need to be dropped.

We cannot now simply check the best rule for each ORF, as we may want several rules to match, for several different classes. So the process essentially remained the same but we look at every matching rule for each ORF, rather than just the best one (finally considering the default class if there are no matching rules).

Writing the confusion matrix has now become awkward because if a prediction was not correct, we could either register it under all the possible classes it should have been (which would heavily skew the numbers) or just one of the classes (which would miss reporting some of the confusions). However, the confusion matrices are reported just for the user, and not used by the program. Since we do not use the confusion matrices in this work they have been left as they are. A new way to represent the results needs to be devised to replace confusion matrices.

Appendix B

Environment, parameters and specifications

Hardware

Most processing was done on 2 Beowulf Linux clusters, "Hrunting" and "Grendel". Hrunting has 20 machines each with an AMD 1.3Ghz processor and 1G memory. Over the course of this project, Grendel has had up to 65 machines with between 128M and 512M memory per node. Grendel's processors are mostly AMD 650MHz chips.

Some processing was done on the department's Sun Enterprise machine which has 20 processors and 10G shared memory. This was used for BLAST jobs that required more memory than the Beowulf clusters could provide, and as a backup when the Beowulf clusters were unavailable.

Copious amounts of filespace on several machines have been used to store databases, PSI-BLAST output, data mining results and backups of everything.

Software

- Decision tree software: C4.5 release 8, used with no options except "-f". Modifications to C4.5 were written in C (compiled with gcc).
- PSI-BLAST: version BLASTP 2.0.12 [Apr-21-2000], options "-e 10 -h 0.0005 -j 20".
- Secondary structure prediction was done by Prof v1.0 (Ouali & King, 2000).
- Expression data clustering: EPClust web based software provided by the EBI at http://ep.ebi.ac.uk/EP/EPCLUST/. QT_CLUST_MOD was my

own software written in Haskell, based on the description given in Heyer *et al.* (1999).

- Multiple sequence alignment: Clustal W was provided by a web based server at the EBI http://www.ebi.ac.uk/clustalw/. This was version 1.82.
- Scripts for results collection, evaluation and statistics were written mostly in Perl 5 with some Haskell.
- Discretisation algorithms (binning and entropy-based) were written in Haskell.
- Graphs in Chapter 8 produced by Matlab 6.
- PolyFARM was written in Haskell98 (using both GHC and NHC98).
- WARMR was part of ACE (we used versions up to 1.1.9) from the Katholieke Universiteit Leuven.
- Aleph was versions 1 and 2.
- Clustering of yeast-yeast PSI-BLAST data to find connected homology clusters was done in Prolog (Yap), as was preprocessing of GeneOntology classification for Chapter 5. Several inconsistencies were found and reported in the GO structure by simply asking `parent(X,X)?`.

Appendix C

Alignment of the mitochondrial carrier family proteins

Here we demonstrate the multiple sequence alignment produced by Clustal W[1], version 1.82, for all ORFs except YMR288W, YHR190W and YMR192W, matching rule 76, level 2 of the structure data. We then overlay the secondary structure predictions made by PROF onto this sequence alignment and a striking pattern appears. Section 8.8 explains the rule and the use of these alignments.

```
CLUSTAL W (1.82) multiple sequence alignment

YJL133W         -----------------------------------MVENSSSNNSTRPIPAIPM----- 19
YKR052C         -------------------------------------MNTSELSIAEEI---------- 12
YIL006W         MTQTDNPVPNCGLLPEQQYCSADHEEPLLLHEEQLIFPDHSSQLSSADIIEPIKMNSSTE 60
YIL134W         ------------------------------------------------------------
YMR166C         ------------------------MNSWNLSSSIPIIHTPHDHPPTSEGTPDQPNNNRK 35
YJR095W         ------------------------------------------------------------
YBR291C         ------------------------------------------------------------
YOR222W         ------------------------------------------------------------
YBR104W         ----------------------------------------MSEEFPTPQLLDELED 16
YOR100C         ----------------------------------------MSSDTSLSESSLLKEES 17
YOR130C         ------------------------------------------------------------
YDL198C         ------------------------------------------------------------
YGR096W         ------------------------------------------------------------
YDL119C         ------------------------------------------------------------
YGR257C         ---------------------MSDRNTSNSLTLKERMLSAGAGSVLTSLILTPMDVVRIR 39
YKL120W         ------------------------------------------------------MSSD 4
YLR348C         ------------------------------------------------------------
YMR056C         ------------------------------------------------------------
YPR128C         ------------------------------------------------------------
```

[1]http://www.ebi.ac.uk/clustalw

```
YJL133W         ----DLPDYEALPTHAPLYHQLIAGAFAGIMEHSVMFPIDALKTRIQSAN---------- 65
YKR052C         -------DYEALPSHAPLHSQLLAGAFAGIMEHSLMFPIDALKTRVQAAG---------- 55
YIL006W         SIIGTTLRKKWVPLSSTQITALSG-AFAGFLSGVAVCPLDVAKTRLQAQGL--------- 110
YIL134W         -----MVDHQWTPLQKEVISGLS----AGSVTTLVVHPLDLLKVRLQLS----------- 40
YMR166C         DDKLHKKRGDSDEDLSPIWHCVVSGGIGGKIGDSAMHSLDTVKTRQQGAP---------- 85
YJR095W         --------MSQKKKASHPAINLMAGGTAGLFEALCCHPLDTIKVRMQIYRR--------- 43
YBR291C         ------MSSKATKSDVDPLHSFLAGSLAGAAEACITYPFEFAKTRLQLID---------- 44
YOR222W         ------MSSDSNAKPLPFIYQFISGAVAGISELTVMYPLDVVKTRFQLEVTTPTA----A 50
YBR104W         QQKVTTPNEKRELSSNRVLKDIFAGTIGGIAQVLVGQPFDTTKVRLQTAT---------- 66
YOR100C         GSLTKSRPPIKSNPVRENIKSFVAGGVGGVCAVFTGHPFDLIKVRCQNGQAN-------- 69
YOR130C         -----MEDSKKKGLIEGAILDIINGSIAGACGKVIEFPFDTVKVRLQTQAS--------- 46
YDL198C         --------MPHTDKKQSGLARLLGSASAGIMEIAVFHPVDTISKRLMSNHT--------- 43
YGR096W         --MFKEEDSLRKGQNVAAWKTLLAGAVSGLLARSITAPMDTIKIRLQLTPAN-------- 50
YDL119C         --------MTEQATKPRNSSHLIGGFFGGLTSAVALQPLDLLKTRIQQDKK--------A 44
YGR257C         LQQQQMIPDCSCDGAAEVPNAVSSGSKMKTFTNVGGQNLNNAKIFWESACFQ--E----L 93
YKL120W         NSKQDKQIEKTAAQKISKFGSFVAGGLAACIAVTVTNPIELIKIRMQLQGEMS------- 57
YLR348C         -----MSTNAKESAGKNIKYPWWYGGAAGIFATMVTHPLDLAKVRLQAAP---------- 45
YMR056C         -----MSHTETQTQQSHFGVDFLMGGVSAAIAKTGAAPIERVKLLMQNQEEMLK------ 49
YPR128C         ---------------MLTLESALTGAVASAMANIAVYPLDLSKTIIQSQVSPSSSEDSNE 45
                                                  .:  .

YJL133W         ---AKSLSAKNMLSQISHISTSEGT-----LALWKGVQSVILGAGPAHAVYFGTYEFCKK 117
YKR052C         ---LNKAASTGMISQISKISTMEGS-----MALWKGVQSVILGAGPAHAVYFGTYEFCKA 107
YIL006W         QTRFENPYYRGIMGTLSTIVRDEGP-----RGLYKGLVPIVLGYFPTWMIYFSVYEFSKK 165
YIL134W         ATSAQKAHYGPFMVIKEIIRSSANSGRSVTNELYRGLSINLFGNAIAWGVYFGLYGVTKE 100
YMR166C         ----NVKKYRNMISAYRTIWLEEGVR----RGLYGGYMAAMLGSFPSAAIFFGTYEYTKR 137
YJR095W         VAGIEHVKPPGFIKTGRTIYQKEGF-----LALYKGLGAVVIGIIPKMAIRFSSYEFYRT 98
YBR291C         ---KASKASRNPLVLIYKTAKTQGIG-----SIYVGCPAFIIGNTAKAGIRFLGFDTIKD 96
YOR222W         AVGKQVERYNGVIDCLKKIVKKEGFS-----RLYRGISSPMLMEAPKRATKFACNDQYQK 105
YBR104W         -------TRTTTLEVLRNLVKNEGVF-----AFYKGALTPLLGVGICVSVQFGVNEAMKR 114
YOR100C         ---STVHAITNIIKEAKTQVKGTLFTN-SVKGFYKGVIPPLLGVTPIFAVSFWGYDVGKK 125
YOR130C         ------NVFPTTWSCIKFTYQNEGIAR----GFFQGIASPLVGACLENATLFVSYNQCSK 96
YDL198C         --KITSGQELNRVIFRDHFSEPLGKR---LFTLFPGLGYAASYKVLQRVYKYGGQPFANE 98
YGR096W         ---GLKPFGSQVMEVARSMIKNEGIR-----SFWKGNIPGSLLYVTYGSAQFSSYSLFNR 102
YDL119C         TLWKNLKEIDSPLQLWRGTLPSALRTS-IGSALYLSCLNLMRSSLAKRRNAVPSLTNDSN 103
YGR257C         HCKNSSLKFNGTLEAFTKIASVEGIT-----SLWRGISLTLLMAIPANMVYFSGYEYIRD 148
YKL120W         --ASAAKVYKNPIQGMAVIFKNEGIK-----GLQKGLNAAYIYQIGLNGSRLGFYEPIRS 110
YLR348C         ------MPKPTLFRMLESILANEGVVG-----LYSGLSAAVLRQCTYTTVRFGAYDLLKE 94
YMR056C         -QGSLDTRYKGILDCFKRTATHEGIVS-----FWRGNTANVLRYFPTQALNFAFKDKIKS 103
YPR128C         GKVLPNRRYKNVVDCMINIFKEKGILG-----LYQGMTVTTVATFVQNFVYFFWYTFIRK 100
                                              :  .

YJL133W         NLIDSSD---------------TQTHHPFKTAISGACATTASDALMN-PFDTIKQRIQLN 161
YKR052C         RLISPED---------------MQTHQPMKTALSGTIATIAADALMN-PFDTVKQRLQLD 151
YIL006W         FFHGIFP-----------------QFDFVAQSCAAITAGAASTTLTN-PIWVVKTRLMLQ 207
YIL134W         LIYKSVAKPG---ETQLKGVGNDHKMNSLIYLSAGASSGLMTAILTN-PIWVIKTRIMST 156
YMR166C         TMIEDWQ-----------------INDTITHLSAGFLGDFISSFVYV-PSEVLKTRLQLQ 179
YJR095W         LLVNKES----------------GIVSTGNTFVAGVGAGITEAVLVVNPMEVVKIRLQAQ 142
YBR291C         MLRDSETG----------------ELSGTRGVIAGLGAGLLESVAAVTPFEAIKTALIDD 140
```

```
YOR222W         IFKNLFN---------------TNETTQKISIAAGASAGMTEAAVIV-PFELIKIRMQDV 149
YBR104W         FFQNYNASKNPNMSSQDVDLSRSNTLPLSQYYVCGLTGGVVNSFLAS-PIEQIRIRLQTQ 173
YOR100C         LVTFNNKQGG------------SNELTMGQMAAAGFISAIPTTLVTA-PTERVKVVLQTS 172
YOR130C         FLEKHTN-----------------VFPLGQILISGGVAGSCASLVLT-PVELVKCKLQVA 138
YDL198C         FLNKHYKKDFDN---------LFGEKTGKAMRSAAAGSLIGIGEIVLLPLDVLKIKRQTN 149
YGR096W         YLTPFGL-----------------EARLHSLVVGAFAGITSSIVSY-PFDVLRTRLVAN 143
YDL119C         IVYNKSSS--------------LPRLTMYENLLTGAFARGLVGYITM-PITVIKVRYEST 148
YGR257C         VSPIAST------------------YPTLNPLFCGAIARVFAATSIA-PLELVKTKLQSI 189
YKL120W         SLNQLFFPDQEP----------HKVQSVGVNVFSGAASGIIGAVIGS-PLFLVKTRLQSY 159
YLR348C         NVIPREQ-----------------LTNMAYLLPCSMFSGAIGGLAGN-FADVVNIRMQND 136
YMR056C         LLSYDRERDG-------------YAKWFAGNLFSGGAAGGLSLLFVY-SLDYARTRLAAD 149
YPR128C         SYMKHKLLGLQSLKNRD----GPITPSTIEELVLGVAAASISQLFTS-PMAVVATRQQTV 155
                                                .  .

YJL133W         TS------------ASVWQTTKQIYQSEG--LAAFYYSYPTTLVMNIPFAAFNFVIYESS 207
YKR052C         TN------------LRVWNVTKQIYQNEG--FAAFYYSYPTTLAMNIPFAAFNFMIYESA 197
YIL006W         SNLG----EHPTHYKGTFDAFRKLFYQEG--FKALYAGLVPSLLG-LFHVAIHFPIYEDL 260
YIL134W         SKGA----QG--AYTSMYNGVQQLLRTDG--FQGLWKGLVPALFG-VSQGALYFAVYDTL 207
YMR166C         GRFNNPFFQSGYNYSNLRNAIKTVIKEEG--FRSLFFGYKATLARDLPFSALQFAFYEKF 237
YJR095W         HLTPSEP-NAGPKYNNAIHAAYTIVKEEG--VSALYRGVSLTAARQATNQGANFTVYSKL 199
YBR291C         KQSATP--KYHNNGRGVVRNYSSLVRDKG--FSGLYRGVLPVSMRQAANQAVRLGCYNKI 196
YOR222W         KS----------SYLGPMDCLKKTIKNEG--IMGLYKGIESTMWRNALWNGGYFGVIYQV 197
YBR104W         TSNG-----GDREFKGPWDCIKKLKAQG-----GLMRGLFPTMIRAGHGLGTYFLVYEAL 223
YOR100C         SK------------GSFIQAAKTIVKEGG--IASLFKGSLATLARDGPGSALYFASYEIS 218
YOR130C         NLQVAS---AKTKHTKVLPTIKAIITERG--LAGLWQGQSGTFIRESFGGVAWFATYEIV 193
YDL198C         PE------------SFKGRGFIKILRDEG--LFNLYRGWGWTAARNAPGSFALFGGNAFA 195
YGR096W         NQMHS---------MSITREVRDIWKLEG--LPGFFKGSIASMTTITLTASIMFGTYETI 192
YDL119C         LYN----------YSSLKEAITHIYTKEG--LFGFFRGFGATCLRDAPYAGLYVLLYEKS 196
YGR257C         PRSSKSTKTWMMVKDLLNETRQEMKMVGP--SRALFKGLEITLWRDVPFSAIYWSSYELC 247
YKL120W         SEFIKIG--EQTHYTGVWNGLVTIFKTEG--VKGLFRGIDAAILRTGAGSSVQLPIYNTA 215
YLR348C         SALEAAK---RRNYKNAIDGVYKIYRYEGG-LKTLFTGWKPNMVRGILMTASQVVTYDVF 192
YMR056C         ARGSKS--TSQRQFNGLLDVYKKTLKTDG--LLGLYRGFVPSVLGIIVYRGLYFGLYDSF 205
YPR128C         HSAES---------AKFTNVIKDIYRENNGDITAFWKGLR-TGLALTINPSITYASFQRL 205
                                                :  .

YJL133W         TKFLN-----------PSNEYNPLIHCLCGSISGSTCAAITTPLDCIKTVLQIRGSQTVS 256
YKR052C         SKFFN-----------PQNSYNPLIHCLCGGISGATCAALTTPLDCIKTVLQVRGSETVS 246
YIL006W         KVRFHCYS--------RENNTNS-INLQRLIMASSVSKMIASAVTYPHEILRTRMQLKSD 311
YIL134W         KQRKLRRK--------RENGLDIHLTNLETIEITSLGKMVSVTLVYPFQLLKS--NLQSF 257
YMR166C         RQLAFKIE--------QKDGRDGELSIPNEILTGACAGGLAGIITTPMDVVKTRVQTQQP 289
YJR095W         KEFLQNYH--------QMDVLPSWETSCIGLISGAIGPFSNAPLDTIKTRLQKDKSISLE 251
YBR291C         KTLIQDY---------TDSPKDKPLSSGLTFLVGAFSGIVTVYSTMPLDTVKTRMQSLDS 247
YOR222W         RNSMP-------------VAKTKGQKTRNDLIAGAIGGTVGTMLNTPFDVVKSRIQSVDA 244
YBR104W         VAREIG-----------TGLTRNEIPPWKLCLFGAFSGTMLWLTVYPLDVVKSIIQNDDL 272
YOR100C         KNYLNSRQPR------QDAGKDEPVNILNVCLAGGIAGMSMWLAVFPIDTIKTKLQASST 272
YOR130C         KKSLKDRHS-------LDDPKRDESKIWELLISGGSAGLAFNASIFPADTVKSVMQTEHI 246
YDL198C         KEYILG------------LKDYSQATWSQNFISSIVGACSSLIVSAPLDVIKTRIQNRNF 243
YGR096W         RIYCDENEK-------TTAAHKKWELATLNHSAGTIGGVIAKIITFPLETIRRRMQFMNS 245
YDL119C         KQLLPMVLPSRFIHYNPEGGFTTYTSTTVNTTSAVLSASLATTVTAPFDTIKTRMQLEPS 256
YGR257C         KERLWLDSTR------FASKDANWVHFINSFASGCISGMIAAICTHPFDVGKTRWQISMM 301
```

```
YKL120W         KNILVKN-------------DLMKDGPALHLTASTISGLGVAVVMNPWDVILTRIYNQKG 262
YLR348C         KNYLVT------------KLDFDASKNYTHLTASLLAGLVATTVCSPADVMKTRIMNGSG 240
YMR056C         KPVLLTG--------------ALEGSFVASFLLGWVITMGASTASYPLDTVRRRMMMTSG 251
YPR128C         KEVFFHDH----------SNDAGSLSAVQNFILGVLSKMISTLVTQPLIVAKAMLQSAGS 255


YJL133W         LEIMRK-----------------ADTFSKAASAIYQVYGWKGFWRGWKPRIVANMPATAI 299
YKR052C         IEIMKD-----------------ANTFGRASRAILEVHGWKGFWRGLKPRIVANIPATAI 289
YIL006W         IPDSIQ-----------------RRLFP-LIKATYAQEGLKGFYSGFTTNLVRTIPASAI 353
YIL134W         RANEQK-----------------FRLFP-LIKLIIANDGFVGLYKGLSANLVRAIPSTCI 299
YMR166C         PSQSNKSYSVTHPHVTNGRPAALSNSISLSLRTVYQSEGVLGFFSGVGPRFVWTSVQSSI 349
YJR095W         KQSGMK-------------------KIITIGAQLLKEEGFRALYKGITPRVMRVAPGQAV 292
YBR291C         TKYSST---------------------MNCFATIFKEEGLKTFWKGATPRLGRLVLSGGI 286
YOR222W         VSSAVKK----------------YNWCLPSLLVIYREEGFRALYKGFVPKVCRLAPGGSL 288
YBR104W         RKPKYKN------------------SISYVAKTIYAKEGIRAFFKGFGPTMVRSAPVNGA 314
YOR100C         RQNMLS----------------------ATKEIYLQRGGIKGFFPGLGPALLRSFPANAA 310
YOR130C         S------------------------LTNAVKKIFGKFGLKGFYRGLGITLFRAVPANAA 281
YDL198C         DNPESG---------------------LRIVKNTLKNEGVTAFFKGLTPKLLTTGPKLVF 282
YGR096W         KHLEKFSRHSSVYG------SYKGYGFARIGLQILKQEGVSSLYRGILVALSKTIPTTFV 299
YDL119C         KFTNSFN----------------------TFTSIVKNENVLKLFSGLSMRLARKAFSAGI 294
YGR257C         NNSDPKG-------------GNRSRNMFKFLETIWRTEGLAALYTGLAARVIKIRPSCAI 348
YKL120W         DLYKGP----------------------IDCLVKTVRIEGVTALYKGFAAQVFRIAPHTIM 301
YLR348C         DHQPALK----------------------ILADAVRKEGPSFMFRGWLPSFTRLGPFTML 278
YMR056C         QTIKYDG-------------------ALDCLRKIVQKEGAYSLFKGCGANIFRGVAAAGV 292
YPR128C         KFTTFQ----------------------EALLYLYKNEGLKSLWKGVLPQLTKGVIVQGL 293
                                                .   :: *    .

YJL133W         SWTAYECAKHFLMTY-------------------- 314
YKR052C         SWTAYECAKHFLMKN-------------------- 304
YIL006W         TLVSFEYFRNRLENISTMVI--------------- 373
YIL134W         TFCVYENLKHRL----------------------- 311
YMR166C         MLLLYQMTLRGLSNAFPTD---------------- 368
YJR095W         TFTVYEYVREHLENLGIFKKNDTPKPKPLK----- 322
YBR291C         VFTIYEKVLVMLA---------------------- 299
YOR222W         MLVVFTGMMNFFRDLKYGH---------------- 307
YBR104W         TFLTFELVMRFLGEE-------------------- 329
YOR100C         TFLGVEMTHSLFKKYGI------------------ 327
YOR130C         VFYIFETLSAL------------------------ 292
YDL198C         SFALAQSLIPRFDNLLSK----------------- 300
YGR096W         SFWGYETAIHYLRMY-------------------- 314
YDL119C         AWGIYEELVKRFM---------------------- 307
YGR257C         MISSYEISKKVFGNKLHQ----------------- 366
YKL120W         CLTFMEQTMKLVYSIESRVLGHN------------ 324
YLR348C         IFFAIEQLKKHRVGMPKEDK--------------- 298
YMR056C         ISLYDQLQLIMFGKKFK------------------ 309
YPR128C         LFAFRGELTKSLKRLIFLYSSFFLKHNGQRKLAST 328
```

The secondary structure is overlaid onto this alignment. The 6 long alpha helices can be seen clearly, each one broken by short coils. conserved glycines and prolines at the locations of these coils are labelled with G and P.

```
YJL133W         ------------------------------------ccccccccccccccaaccc----- 19
YKR052C         ------------------------------------ccccccccccccc---------- 12
YIL006W         ccccccccccccccccccccccccaaaaaaccccccccaaaaaaaaaaaaaaaaccccaa 60
YIL134W         ------------------------------------------------------------
YMR166C         ------------------------cccccccaaaaaaaaccaaaccccccccaaaaaacaa 35
YJR095W         ------------------------------------------------------------
YBR291C         ------------------------------------------------------------
YOR222W         ------------------------------------------------------------
YBR104W         ---------------------------------------------ccccccccccccccaaa 16
YOR100C         ---------------------------------------------ccccccccaaaaaaaaa 17
YOR130C         ------------------------------------------------------------
YDL198C         ------------------------------------------------------------
YGR096W         ------------------------------------------------------------
YDL119C         ------------------------------------------------------------
YGR257C         ---------------------cccccccccccccaaaaaaaaaaaaaaaaaaaccaaaaaaa 39
YKL120W         ------------------------------------------------------cccc 4
YLR348C         ------------------------------------------------------------
YMR056C         ------------------------------------------------------------
YPR128C         ------------------------------------------------------------


YJL133W         ----ccccccccccccaaaaaaaaaaaaaaaaaaaaccccaaaaaaaaaa---------- 65
YKR052C         -------ccccccccccaaaaaaaaaaaaaaaaaaccaaaaaaaaaaacc---------- 55
YIL006W         cacccccccccccccccaaaaaaa-aaaaaaaaaaaccaaaaaaaaaaccc--------- 110
YIL134W         -----cccccccaaaaaaaaaa----aaaaaaaaaccaaaaaaaaaac----------- 40
YMR166C         aaaaaaaccccccccccaaaaaaaaaaaaaaaaaaaccaaaaaaaaaccc---------- 85
YJR095W         --------ccccccccaaaaaaaaaaaaaaaaaaaccaaaaaaaaaaccc--------- 43
YBR291C         ------ccccccccccaaaaaaaaaaaaaaaaaaaccaaaaaaaaaacc---------- 44
YOR222W         ------ccccccccccaaaaaaaaaaaaaaaaaaaccaaaaaaaaaaccccccc----c 50
YBR104W         ccccccccccccccccaaaaaaaaaaaaaaaaaaacccaaaaaaaaaaa---------- 66
YOR100C         aaaccccccccccccaaaaaaaaacaaaaaaaabbbccaaaaaaaaaaacccc-------- 69
YOR130C         -----ccccccccccaaaaaaaaaaaaaaaaaaaaccaaaaaaaaaaccc--------- 46
YDL198C         --------ccccccccaaaaaaaaaaaaaaaaaaaccaaaaaaaaaaccc--------- 43
YGR096W         --ccaaaccccccccccaaaaaaaaaaaaaaaaaaaccaaaaaaaaaacccc-------- 50
YDL119C         --------ccccccccccaaaaaaaaaaaaaaaaaaacccaaaaaaaaaaccc--------c 44
YGR257C         aaaccccccccccccccccaaaaaaaaaaaaaaccccccacaaaaaccccccc--c----c 93
YKL120W         accccccccccccccccccaaaaaaaaaaaaaaaaaaccaaaaaaaaaacccccc------- 57
YLR348C         -----ccccccccccccaaaaaaaaaaaaaaaaaacccaaaaaaaaaacc---------- 45
YMR056C         -----ccccccccccccaaaaaaaaaaaaaaaaaaaccaaaaaaaaaacccccc------ 49
YPR128C         ---------------caaaaaaaaaaaaaaaaaaaccaaaaaaaaaacccccccccccc 45
                                              P.:  .

YJL133W         ---aacaacccaaaaaaaaaaaccc-----aaaaccccaaaaaaaaaaaaaaaaaaaaaa 117
YKR052C         ---cccccccaaaaaaaaaaaccc-----caaacccaaaaaaaaaaaaaaaaaaaaaaa 107
YIL006W         ccccccccccccaaaaaaaaaaaccc-----caaacccaaaaaaaaaaaaaaaaaaaaaa 165
YIL134W         ccccccccccccaaaaaaaaaaccccccccaaaaaaaacccaaaaaaaaaaaaaaaaaaa 100
```

```
YMR166C         ----ccccccccaaaaaaaaaaaccccc----aaaaacccaaaaaaaaaaaaaaaaaaaaaa 137
YJR095W         ccccccccccccaaaaaaaaaaaccc-----aaaacccaaaaaaaaaaaaaaaaaaaaaaa 98
YBR291C         ---ccccccccaaaaaaaaaaaccccc-----aaccccaaaaaaaaaaaaaaaacaaaaaa 96
YOR222W         ccccccccccccaaaaaaaaaaaccccc-----caacccaaaaaaaaaaaaaaaaaaaaaa 105
YBR104W         -------aaaaaaaaaaaaaaaccca-----aaaccccaaaaaaaaaaaaaaaaaaaaaa 114
YOR100C         ---ccccccccccccaaaaaaaaaaac-cccaaaccccaaaaaaaaaaaaaaaaaaaaaa 125
YOR130C         ------ccccaaaaaaaaaaaaccccca----aaaccccaaaaaaaaaaaaaaaaaaaaa 96
YDL198C         --ccccccccaaaaaaaaaaaaaacc---caaaaccccaaaaaaaaaaaaaccaaaaaaa 98
YGR096W         ---ccccccccaaaaaaaaaaaccccc-----aaccccaaaaaaaaaaaaaaaaaaaaaa 102
YDL119C         aaaaaaaaaccccccccccaaaaaaaa-aaaaaaaaaaaaaaaaaaccccccccccccc 103
YGR257C         ccccccccccaaaaaaaaaaaccca-----aaaccccaaaaaaaaaaaaaaaaaaaaaa 148
YKL120W         --ccccccccaaaaaaaaaaaccccc-----aaaccccaaaaaaaaaaaaaaaaaaaaaa 110
YLR348C         ------cccccaaaaaaaaaaaccccaa-----cccccaaaaaaaaaaaaaaaaaaaaaa 94
YMR056C         -ccccccccccaaaaaaaaaaaccccaa-----aaccccaaaaaaaaaaaaaaaaaaaaaa 103
YPR128C         ccccccccccaaaaaaaaaaaccccaa-----aaccccaaaaaaaaaaaaaaaaaaaaaa 100
                                  G        :  G

YJL133W         aaccccc---------------ccccaaaaaaaaaaaaaaaaaaac-caaaaaaaaaac 161
YKR052C         aaccccc---------------ccccaaaaaaaaaaaaaaaaaacc-caaaaaaaaaac 151
YIL006W         aaccccc-----------------ccccaaaaaaaaaaaaaaaaac-caaaaaaaaaac 207
YIL134W         aaaaccccccc---ccccccccccccccaaaaaaaaaaaaaaaaac-caaaaaaaaaac 156
YMR166C         aaaccccc-----------------ccccaaaaaaaaaaaaaaaaac-caaaaaaaaaac 179
YJR095W         aaaccccc----------------ccccaaaaaaaaaaaaaaaaaacccaaaaaaaaaac 142
YBR291C         aaaccccc----------------cccaaaaaaaaaaaaaaaaaaccaaaaaaaaaac 140
YOR222W         aaccccc---------------ccccccaaaaaaaaaaaabbbcc-caaaaaaaaaac 149
YBR104W         aaaccccccccccaccccccccccccaaaaaaaaaaaaaaaaaac-caaaaaaaaaac 173
YOR100C         aaaccccccc------------cccccaaaaaaaaaaaaaaaaac-caaaaaaaaaac 172
YOR130C         aaaccccc-----------------ccccaaaaaaaaaaaaaaaac-caaaaaaaaaac 138
YDL198C         aaaaaaccccccc---------ccccccccaaaaaaaaaaaaaaaaaccaaaaaaaaac 149
YGR096W         aaccccc-----------------ccccaaaaaaaaaaaaaaaac-caaaaaaaaaac 143
YDL119C         cccccccc--------------cccccaaaaaaaaaaaaaaaaac-caaaaaaaaaac 148
YGR257C         aaccccc-------------------ccccaaaaaaaaaaaaaaac-caaaaaaaaaac 189
YKL120W         aaaaccccccccc----------ccccaaaaaaaaaaaaaaaaaccc-caaaaaaaaac 159
YLR348C         aaccccc-----------------cccaaaaaaaaaaaaaaaaac-caaaaaaaaaac 136
YMR056C         aaaccccccc-------------ccccaaaaaaaaaaaaaaaccccc-caaaaaaaaac 149
YPR128C         aaaacccccccccccc----ccccaaaaaaaaaaaaaaaaaaac-caaaaaaaaac 155
                                          .  .          P

YJL133W         cc------------ccaaaaaaaaaacc--caaaacccaaaaaaaccaaaaaaaaaaa 207
YKR052C         cc------------cccaaaaaaaaacc--caaaacccaaaaaaaccaaacaaaaaaa 197
YIL006W         cccc----ccccccccaaaaaaaaaacc--caaaacccaaaaaa-aaaaacaaaaaaa 260
YIL134W         cccc----cc--ccccaaaaaaaaaacc--ccaaacccaaaaaa-aaaaaaaaaaaaa 207
YMR166C         ccccccccccccccccaaaaaaaaaacc--caaaaacaaaaaaaaccaaacaaaaaaa 237
YJR095W         ccccccc-cccccccaaaaaaaaaacc--caaaacccaaaaaaaccaaaaaaaaaaa 199
YBR291C         cccccc--cccccccaaaaaaaaaacc--caaaacccaaaaaaacaaaaaaaaaaaa 196
YOR222W         cc----------ccccaaaaaaaaaacc--caaaacccaaaaaaaaaaaaaaaaaaa 197
YBR104W         cccc-----cccccccaaaaaaaaaac-----caaccccaaaaaaaaaacaaaaaaa 223
YOR100C         cc------------ccaaaaaaaaaacc--caaaaccccaaaaaaccaaaaaaaaaa 218
YOR130C         cccccc---cccccccaaaaaaaaaacc--caaaacccaaaaaaaaaaaacaaaaaaa 193
YDL198C         cc------------ccccaaaaaaaacc--caaaacccaaaaaaaaaaaaaaaaaaa 195
```

```
YGR096W         ccccc---------ccaaaaaaaaaaacc--caaaacccaaaaaaaaaaaaaaaaaaaa 192
YDL119C         ccc----------ccaaaaaaaaaaaacc--caaaacccaaaaaaacaaaaaaaaaaaa 196
YGR257C         ccccccccccccaaaaaaaaaaaaaccc--caaaaccccaaaaaaccaaaccaaaaaaa 247
YKL120W         ccccccc--cccccccaaaaaaaaaaacc--caaaaccccaaaaaaaaaaaaaaaaaaa 215
YLR348C         ccccccc---cccccaaaaaaaaaaaccc-aaaaaccccaaaaaaaaaaaaaaaaaaa 192
YMR056C         cccccc--cccccccaaaaaaaaaaacc--ccaaacccaaaaaaaaaaaaaaaaaaaa 205
YPR128C         ccccc---------cccaaaaaaaaaaacccaaaaaccca-aaaaaaaacccaaaaaaaaa 205
                                        G     :  G

YJL133W         aaaac-----------ccccccaaaaaaaaaaaaaaaaacccaaaaaaaaaaccccccc 256
YKR052C         aaaac-----------ccccccaaaaaaaaaaaaaaaaacccaaaaaaaaaaccccccc 246
YIL006W         aaaaaacc--------ccccccc-cccaaaaaaaaaaaaaaaacccaaaaaaaaaaccc 311
YIL134W         aaaaaccc--------ccccccccccaaaaaaaaaaaaaaaaacccaaaaaa--aaaac 257
YMR166C         aaaaaacc--------ccccccccccaaaaaaaaaaaaaaaaacccaaaaaaaaaaccc 289
YJR095W         aaaacccc--------ccccccaaaaaaaaaaaaaaaaacccaaaaaaaaaccccccc 251
YBR291C         aaaaaac---------ccccccccccaaaaaaaaaaaaaaaaacccaaaaaaaaaaccc 247
YOR222W         aaaaa-------------ccccccccaaaaaaaaaaaaaaaaacccaaaaaaaaaaccc 244
YBR104W         aaaaac-----------ccccccccccaaaaaaaaaaaaaaaaacccaaaaaaaaaaccc 272
YOR100C         aaaaaacccc------ccccccccccaaaaaaaaaaaaaaaaacccaaaaaaaaaaccc 272
YOR130C         aaaaaaccc-------ccccccccccaaaaaaaaaaaaaaaaacccaaaaaaaaaacccc 246
YDL198C         aaaaac-----------ccccccccaaaaaaaaaaaaaaaaacccaaaaaaaaaaccc 243
YGR096W         aaaaacccc-------ccccccccccbaaaaaaaaaaaaaaaacccaaaaaaaaaaccc 245
YDL119C         aaaaaaccccccccccccccccccaaaaaaaaaaaaaaaaaacccaaaaaaaaaaccc 256
YGR257C         aaaaaacccc------ccccccccccaaaaaaaaaaaaaaaaacccaaaaaaaaaaccc 301
YKL120W         aaaaccc-------------ccccccaaaaaaaaaaaaaaaacccaaaaaaaaaaccc 262
YLR348C         aaaaaa------------ccccccccaaaaaaaaaaaaaaaacccaaaaaaaaaaccc 240
YMR056C         aaaaccc--------------ccccccaaaaaaaaaaaaaaaacccaaaaaaaaaaccc 251
YPR128C         aaaaaccc----------ccccccccaaaaaaaaaaaaaaaacccaaaaaaaaaaccc 255
                                                    P

YJL133W         cccccc-----------------cccaaaaaaaaaaacccccaaaaccaaaaaaacaaaaa 299
YKR052C         cccccc-----------------cccaaaaaaaaaaacccaaaaaccaaaaaaacccaaaa 289
YIL006W         cccccc-----------------ccaaa-aaaaaaaaccccaaacccaaaaaaaccaaaa 353
YIL134W         cccccc-----------------ccaaa-aaaaaaaaccccaaaaaccaaaaaaaccaaaa 299
YMR166C         ccccccccccccccccccccccccaaaaaaaaaaaacccaaaaaccaaaaaaaccaaaa 349
YJR095W         cccccc-------------------caaaaaaaaaaacccaaaaaccaaaaaaaccaaaa 292
YBR291C         ccccca---------------------aaaaaaaaaacccaaaaaccaaaaaaaccaaaa 286
YOR222W         ccccccc-----------------cccaaaaaaaaaaacccaaaaaccaaaaaaacaaaaa 288
YBR104W         ccccccc-------------------caaaaaaaaaaacccaaaaaccaaaaaaaccaaaa 314
YOR100C         ccccaa----------------------aaaaaaaaacccaaaaaccaaaaaaaccaaaa 310
YOR130C         c-------------------------aaaaaaaaaaaccccaaacccaaaaaaaccaaaa 281
YDL198C         ccccca----------------------aaaaaaaaaacccaaaaaccaaaaaaaaaaaa 282
YGR096W         ccccccccccccc------ccccccaaaaaaaaaaaacccaaaaaccaaaaaaaccaaaa 299
YDL119C         ccccaaa---------------------aaaaaaaacccaaaaaccaaaaaaccaaaa 294
YGR257C         ccccccc-------------ccccccaaaaaaaaaaaacccaaaaaccaaaaaaaccaaaa 348
YKL120W         ccccca----------------------aaaaaaaaaacccaaaaaccaaaaaaaccaaaa 301
YLR348C         ccccaaa---------------------aaaaaaaacccaaaaaccaaaaaaaccaaaa 278
YMR056C         ccccccc-------------------aaaaaaaaaaacccaaaaaccaaaaaaaccaaaa 292
YPR128C         ccccaa----------------------aaaaaaaaacccaaaaaccaaaaaaaccaaaa 293
                                                  G   :: G    .
```

```
YJL133W         aaaaaaaaaaaaacc------------------- 314
YKR052C         aaaaaaaaaaaaacc------------------- 304
YIL006W         aaaaaaaaaaaaaaaaaacc--------------- 373
YIL134W         aaaaaaaaaaac---------------------- 311
YMR166C         aaaaaaaaaaaaaaccccc---------------- 368
YJR095W         aaaaaaaaaaaaaaccccccccccccccc----- 322
YBR291C         aaaaaaaaaaaac--------------------- 299
YOR222W         aaaaaaaaaaaaaccccccc--------------- 307
YBR104W         aaaaaaaaaaaaacc------------------- 329
YOR100C         aaaaaaaaaaaaaaacc----------------- 327
YOR130C         aaaaaaaaaac----------------------- 292
YDL198C         aaaaaaaaaaaaaaaccc---------------- 300
YGR096W         aaaaaaaaaaaaacc------------------- 314
YDL119C         aaaaaaaaaaaac--------------------- 307
YGR257C         aaaaaaaaaaaaaaaccc---------------- 366
YKL120W         aaaaaaaaaaaaaaccccccccc----------- 324
YLR348C         aaaaaaaaaaaaccccccccc------------- 298
YMR056C         aaaaaaaaaaaaaaccc----------------- 309
YPR128C         aaaaaaaaaaaaaaaaccccccccccccccccccc 328
```

References

Agrawal, R., & Shafer, J. 1996. Parallel mining of assocation rules. *IEEE Trans. on Knowledge and Data Engineering*, **8(6)**(Dec), 962–969.

Agrawal, R., & Srikant, R. 1994. Fast algorithms for mining association rules in large databases. *In: 20th International Conference on Very Large Databases (VLDB 94)*. Expanded version: IBM Research Report RJ9839, June 1994.

Agrawal, R., Imielinski, T., & Swami, A. 1993. Database mining: a performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, **5(6)**, 914–925.

Aha, D. W., & Bankert, R. L. 1995. A Comparative evaluation of Sequential Feature Selection Algorithms. *Pages 1–7 of: Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics*.

Alba, M. M., Santibáñez-Koref, M. F., & Hancock, J. M. 1999. Amino Acid Reiterations in Yeast are Overrepresented in Particular Classes of Proteins and Show Evidence of a Slippage-like Mutational Process. *Journal of Molecular Evolution*, **49**, 789–797.

Ali, K., & Pazzani, M. 1996. Error reduction through learning multiple descriptions. *Machine Learning*, **24(3)**.

Almuallim, H., Akiba, Y., & Kaneda, S. 1995. On handling tree-structured attributes in decision tree learning. *In: Proceedings of the 12th International Conference on Machine Learning (ICML95)*.

Almuallim, H., Akiba, Y., & Kaneda, S. 1997. An Efficient Algorithm for Finding Optimal Gain-Ratio Multiple-Split Tests on Hierarchical Attributes in Decision Tree Learning. *In: Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*.

Alon, U., Barkai, N., Notterman, D., Gish, K., Ybarra, S., Mack, D., & Levine, A. 1999. Broad patterns of gene expression revealed by clustering analysis of

tumor and normal colon tissues probed by oligonucleotide arrays. *Proc. Nat. Acad. Sci. USA*, **96(12)**(Jun), 6745–50.

Altschul, S. F., Madden, T. L., Schaffer, A. A., Zhang, J., Zhang, Z., Miller, W., & Lipman, D. J. 1997. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucl. Acids. Res*, **25**, 3389–3402.

Andersen, P. 2001. TB vaccines: progress and problems. *Trends in Immunology*, **22(3)**, 160–168.

Andrade, M., Brown, N. P., Leroy, C., Hoersch, S., de Daruvar, A., Reich, C., Franchini, A., Tamames, J., Valencia, A., Ouzounis, C., & Sander, C. 1999. Automated genome sequence analysis and annotation. *Bioinformatics*, **15(5)**, 391–412.

Arabidopsis genome initiative, The. 2000. Analysis of the genome sequence of the flowering plant Arabidopsis thaliana. *Nature*, **408**, 796–815.

Attwood, T.K., Blythe, M., Flower, D.R., Gaulton, A., Mabey, J.E., Maudling, N., McGregor, L., Mitchell, A., Moulton, G., Paine, K., & Scordis, P. 2002. PRINTS and PRINTS-S shed light on protein ancestry. *Nucleic Acids Research*, **30(1)**, 239–241.

Badea, L. 2000. Learning Trading Rules with Inductive Logic Programming. *In: 11th European Conference on Machine Learning.*

Bairoch, A., Bucher, P., & Hofmann, K. 1996. The PROSITE database, its status in 1995. *Nucleic Acids Research.*

Baker, P.G., Goble, C.A., Bechhofer, S., Paton, N.W., Stevens, R., & Brass, A. 1999. An Ontology for Bioinformatics Applications. *Bioinformatics*, **15(6)**, 510–520.

Barash, Y., & Friedman, N. 2001. Context-Specific Bayesian Clustering for Gene Expression Data. *In: Proceedings of the Fifth International Conference on Computational Molecular Biology (RECOMB 2001).*

Baudinet, M., Chomicki, J., & Wolper, P. 1993. *Temporal Deductive Databases.*

Bauer, E., & Kohavi, R. 1999. An empirical comparison of voting classification algorithms: bagging, boosting and variants. *Machine Learning*, **36**, 105–139.

Baxevanis, A. D. 2002. The Molecular Biology Database Collection: 2002 update. *Nucleic Acids Research*, **30(1)**, 1–12.

Behr, M. A., Wilson, M. A., Gill, W. P., Salamon, H., Schoolnik, G. K., Rane, S., & Small, P. M. 1999. Comparative genomics of BCG vaccines by whole-genome DNA microarray. *Science*, **284**, 1520–1523.

Bender, R., & Lange, S. 1999. Multiple test procedures other than Bonferroni's deserve wider use. *BMJ*, **318**, 600.

Blake, C.L., & Merz, C.J. 1998. *UCI Repository of machine learning databases.*

Blat'ák, J., Popelínský, L., & Nepil, M. 2002. RAP: Framework for mining maximal frequent Datalog queries. *In: First International Workshop on Knowledge Discovery in Inductive Databases (KDID'02).*

Blattner, F.R., Plunkett, G., Bloch, C. A., Perna, N. T., Burland, V., Riley, M., Collado-Vides, J., Glasner, J. D., Rode, C. K., Mayhew, G. F., Gregor, J., Davis, N. W., Kirkpatrick, H. A., Goeden, M. A., Rose, D. J., Mau, B., & Shao, Y. 1998. The complete genome sequence of Escherichia coli K-12. *Science*, **277(5331)**, 1453–74.

Blockeel, H., & De Raedt, L. 1998. Top-down Induction of First-order Logical Decision Trees. *Artificial Intelligence*, **101(1-2)**, 285–297.

Blockeel, H., De Raedt, L., & Ramon, J. 1998. Top-down Induction of Clustering Trees. *Pages 55–63 of: 15th International Conference on Machine Learning (ICML 98).*

Blockeel, H., Bruynooghe, M., Dzeroski, S., Ramon, J., & Struyf, J. 2002 (July). Hierarchical multi-classification. *Pages 21–35 of:* Dzeroski, S., De Raedt, L., & S., Wrobel (eds), *Proceedings of the Workshop on Multi-Relational Data Mining (MRDM-2002).*

Bock, J. R., & Gough, D. A. 2001. Predicting protein-protein interactions from primary structure. *Bioinformatics*, **17**, 455–460.

Bork, P., Dandekar, T., Diaz-Lazcoz, Y., Eisenhaber, F., Huynen, M., & Yuan, Y. 1998. Predicting Function: From Genes to Genomes and Back. *Journal of Molecular Biology*, **283**, 707–725.

Bostrom, Henrik. 1998. Predicate Invention and Learning from Positive Examples Only. *Pages 226–237 of: European Conference on Machine Learning.*

Boucherie, H., Dujardin, G., Kermorgant, M., Monribot, C., Slonimski, P., & Perrot, M. 1995. Two dimensional protein map of *Saccharomyces cerevisiae*: construction of a gene-protein index. *Yeast*, **11**, 601–613.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. 1984. *Classification and Regression Trees.* Pacific Grove: Wadsworth.

Brenner, S. 1999. Errors in Genome Annotation. *Trends Genet.*, **15(4)**(April), 132–133.

Brown, M., Nobel Grundy, W., Lin, D., Cristianini, N., Walsh Sugnet, C., Furey, T., Ares Jr., M., & Haussler, D. 2000. Knowledge-based analysis of microarray gene expression data by using support vector machines. *Proc. Nat. Acad. Sci. USA*, **97(1)**(Jan), 262–267.

Burns, N., Grimwade, B., Ross-Macdonald, P. B., Choi, E. Y., Finberg, K., Roeder, G. S., & Snyder, M. 1994. Large-scale analysis of gene expression, protein localisation, and gene disruption in *Saccharomyces cerevisiae. Genes Dev.*, **8**, 1087–1105.

Butte, A., & Kohane, I. 2000. Mutual information relevance networks: functional genomic clustering using pairwise entropy measurements. *In: PSB 2000.*

CASP. 2001. *Fourth Meeting on the Critical Assessment of Techniques for Protein Structure Prediction.* Supplement in Proteins: Structure, Function and Genetics 45(S5).

Cestnik, B. 1990. Estimating Probabilities: A Crucial Task in Machine Learning. *Pages 147–149 of: Proceedings of the Ninth European Conference on Artificial Intelligence (ECAI90).*

Chakrabarti, S., Dom, B., Agrawal, R., & Raghavan, P. 1998. Scalable feature selection, classification and signature generation for organising large text databases into hierarchical topic taxonomies. *VLDB Journal*, **7**, 163–178.

Cheung, D., Ng, V., Fu, A., & Fu, Y. 1996. Efficient mining of assocation rules in distributed databases. *IEEE Trans. on Knowledge and Data Engineering*, **8(6)**(Dec), 911–922.

Cho, R., Campbell, M., Winzeler, E., Steinmetz, L., Conway, A., Wodicka, L., Wolfsberg, T., Gabrielian, A., Landsman, D., Lockhart, D., & Davis, R. 1998. A genome-wide transcription analysis of the mitotic cell cycle. *Molecular Cell*, **2**(July), 65–73.

Chu, S., DeRisi, J., Eisen, M., Mulholland, J., Botstein, D., Brown, P., & Herskowitz, I. 1998. The transcriptional program of sporulation in budding yeast. *Science*, **282**(Oct), 699–705.

Clare, A., & King, R. D. 2002a. How well do we understand the clusters found in microarray data? *In Silico Biology*, **2**, 0046.

Clare, A., & King, R. D. 2002b. Machine learning of functional class from phenotype data. *Bioinformatics*, **18(1)**, 160–166.

Clare, A., & King, R.D. 2001. Knowledge Discovery in Multi-Label Phenotype Data. *In: Proceedings of 5th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD 2001).*

Cleary, J., Legg, S., & Witten, I. 1996. An MDL estimate of the significance of rules. *Pages 43–53 of: Proceedings of the Information, Statistics and Induction in Science.*

Cole, S. T. 1998. Comparative mycobacterial genomics. *Current Opinion in Microbiology*, **1**, 567–571.

Cole, S.T., Brosch, R., Parkhill, J., Garnier, T., Churcher, C., Harris, D., Gordon, S.V., Eiglmeier, K., Gas, S., Barry, C.E.III., Tekaia, F., Badcock, K., Basham, D., Brown, D., Chillingworth, T., Connor, R., Davies, R., Devlin, K., Feltwell, T., Gentles, S., Hamlin, N., Holroyd, S., Hornsby, T., Jagels, K., Krogh, A., McLean, J., Moule, S., Murphy, L., Oliver, K., Osborne, J., Quail, M.A., Rajandream, M-A., Rogers, J., Rutter, S., Seeger, K., Skelton, J., Squares, S., Squares, R., Sulston, J.E., Taylor, K., Whitehead, S., & Barrell, B.G. 1998. Deciphering the biology of *Mycobacterium tuberculosis* from the complete genome sequence. *Nature*, **393**(June), 537–544.

Cristianini, N., & Shawe-Taylor, J. 2000. *An Introduction to Support Vector Machines*. Cambridge University Press, Cambridge, UK.

Csuros, M. 2001. Fast recovery of evolutionary trees with thousands of nodes. *In: RECOMB 01.*

De Raedt, L., & Dehaspe, L. 1997. Clausal discovery. *Machine Learning*, **26**, 99–146.

Dehaspe, L. 1998. *Frequent Pattern Discovery in First Order Logic*. Ph.D. thesis, Department of Computer Science, Katholieke Universiteit Leuven.

Dehaspe, L., & De Raedt, L. 1997. Mining Association Rules in Multiple Relations. *In: 7th International Workshop on Inductive Logic Programming.*

DeRisi, J., Iyer, V., & Brown, P. 1997. Exploring the Metabolic and Genetic Control of Gene Expression on a Genomic Scale. *Science*, **278**(October), 680–686.

des Jardins, M., Karp, P., Krummenacker, M., Lee, T., & Ouzounis, C. 1997. Prediction of Enzyme Classification from Protein Sequence without the use of Sequence Similarity. *In: ISMB '97.*

D'Haeseleer, P., Liang, S., & Somogyi, R. 2000. Genetic network inference: From co-expression clustering to reverse engineering. *Bioinformatics*, **16(8)**, 707–726.

Dietterich, T.G. 2000. Ensemble methods in machine learning. *In: Proceedings of First International Workshop on Multiple Classifier Systems (MCS 2000).*

Ding, C., & Dubchak, I. 2001. Multi-class protein fold recognition using support vector machines and neural networks. *Bioinformatics*, **17**, 349–358.

Domingues, F.S., Koppensteiner, W. A., & Sippl, M.J. 2000. The role of protein structure in genomics. *FEBS Lett.*, **476**, 98–102.

Duda, R., Hart, P., & Stork, P. 2000. *Pattern Classification.* John Wiley and Sons, New York.

Efron, B., & Tibshirani, R. 1993. *An introduction to the bootstrap.* Chapman and Hall.

Eiben, A.E. 2002. Evolutionary computing: the most powerful problem solver in the universe? *Dutch Mathematical Archive (Nederlands Archief voor Wiskunde)*, **5/3**, 126–131.

Eisen, M., Spellman, P., Brown, P., & Botstein, D. 1998. Cluster analysis and display of genome-wide expression patterns. *Proc. Nat. Acad. Sci. USA*, **95**(Dec), 14863–14868.

Eisenhaber, F., & Bork, P. 1999. Evaluation of human-readable annotation in biomolecular sequence databases with biological rule libraries. *Bioinformatics*, **15(7/8)**, 528–535.

Elomaa, T. 1994. In Defense of C4.5: Notes on Learning One-Level Decision Trees. *Pages 62–69 of: Proceedings 11th Intl. Conf. Machine Learning.* Morgan Kaufmann.

Fasulo, D. 1999. *An analysis of recent work on clustering algorithms.* Tech. rept. 01-03-02. University of Washington.

Fayyad, U., & Irani, K. 1993. Multi-interval discretization of continuous-valued attributes for classification learning. *Pages 1022–1027 of: Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence.*

Feise, R. J. 2002. Do multiple outcome measures require p-value adjustment? *BMC Medical Research Methodology*, **2**, 8.

Feng, C. 1991. Inducing Temporal Fault Diagnostic Rules from a Qualitative Model. *In: Proceedings of the Eighth International Workshop on Machine Learning (ML91).*

Fogel, D.B. 1995. *Evolutionary Computation - Towards a New Philosophy of Machine Intelligence.* IEEE Press, New York.

Freitas, A. 2000. Understanding the Crucial Differences Between Classification and Discovery of Association Rules - A Position Paper. *SIGKDD Explorations*, **2**(1), 65–69.

Fromont-Racine, M., Rain, J., & Legrain, P. 1997. Toward a functional analysis of the yeast genome through exhaustive two-hybrid screens. *Nature Genetics*, **16**, 277–282.

Fukuda, K., Tsunoda, T., Tamura, A., & Tagaki, T. 1998. Toward Information Extraction: Identifying protein names from biological papers. *In: Pacific Symposium in Biocomputing 98.*

Gasch, A., Spellman, P., Kao, C., Carmel-Harel, O., Eisen, M., Storz, G., Botstein, D., & Brown, P. 2000. Genomic expression program in the response of yeast cells to environmental changes. *Mol. Biol. Cell.*, **11**(Dec), 4241–4257.

Gasch, A., Huang, M., Metzner, S., Botstein, D., Elledge, S., & Brown, P. 2001. Genomic expression responses to DNA-damaging agents and the regulatory role of the yeast ATR homolog Mec1p. *Mol. Biol. Cell*, **12**(10), 2987–3000.

Gene Ontology Consortium, The. 2000. Gene Ontology: tool for the unification of biology. *Nature Genet.*, **25**, 25–29.

Gerhold, D., Rushmore, T., & Caskey, C. T. 1999. DNA chips:promising toys have become powerful tools. *Trends in Biochemical Sciences*, **24(5)**, 168–173.

Gershon, D. 2002. Microarray technology: an array of opportunities. *Nature*, **416**, 885–891.

Goff, S. A., *et al.*. 2002. A draft sequence of the rice genome (*Oryza sativa* L. ssp. *japonica*). *Science*, **296**, 92–100.

Goffeau, A., Nakai, K., Slominski, P., & Risler, J. L. 1993. The Membrane Proteins Encoded by Yeast Chromosome III Genes. *FEBS Letters*, **325**, 112–117.

Goffeau, A., Barrell., B., Bussey, H., Davis, R., Dujon, B., Feldmann, H., Galibert, F., Hoheisel, J., Jacq, C., Johnston, M., Louis, E., Mewes, H., Murakami, Y., Philippsen, P., Tettelin, H., & Oliver, S. 1996. Life with 6000 genes. *Science*, **274**, 563–7.

Grosse, I., Buldyrev, S., & Stanley, E. 2000. Average mutual information of coding and non-coding DNA. *In: Pacific Symposium in Biocomputing 2000.*

Guermeur, Y., Geourjon, C., Gallinari, P., & Deleage, G. 1999. Improved Performance in Protein Secondary Structure Prediction by Inhomogeneous Score Combination. *Bioinformatics*, **15(5)**, 413–421.

Han, E., Karypis, G., & Kumar, V. 1997. Scalable parallel data mining for association rules. *In: SIGMOD '97.*

Hanisch, D., Zien, A., Zimmer, R., & Lengauer, T. 2002. Co-clustering of biological networks and gene expression data. *Bioinformatics*, **18**, S145–S154.

Heyer, L. J., Kruglyak, S., & Yooseph, S. 1999. Exploring expression data: identification and analysis of coexpressed genes. *Genome Research*, **9(11)**(Nov), 1106–15.

Higgins, D. G., J.D. Thompson, J. D., & Gibson, T. J. 1994. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position specific gap penalties and weight matrix choice. *Nucleic Acids Research*, **22**, 4673–4680.

Hipp, J., Güntzer, U., & Nakhaeizadeh, G. 2000. Algorithms for Association Rule Mining – A General Survey and Comparison. *SIGKDD Explorations*, **2**(1), 58–64.

Hodges, P., McKee, A., Davis, B., Payne, W., & Garrels, J. 1999. The Yeast Proteome Database (YPD): a model for the organization and presentation of genome-wide functional data. *Nucleic Acids Research*, **27**, 69–73.

Holm, L., & Sander, C. 1998. Removing near-neighbour redundancy from large protein sequence collections. *Bioinformatics*, **14**, 423–429.

Horton, P., & Nakai, K. 1996. A Probabilistic Classification System for Predicting the Cellular Localisation Sites of Proteins. *Pages 109–115 of: ISMB '96.*

Hudak, J., & McClure, M.A. 1999. A Comparative Analysis of Computational Motif-Detection Methods. *Pages 138–149 of: Pacific Symposium on Biocomputing.*

Humphreys, K., Demetriou, G., & Gaizauskas, R. 2000. Two applications of Information Extraction to Biological Science Journal Articles: Enzyme Interactions and Protein Structures. *In: Pacific Symposium on Biocomputing 2000.*

Hvidsten, T.R., Komorowski, J., Sandvik, A.K., & Lægreid, A. 2001. Predicting Gene Function from Gene Expressions and Ontologies. *Pages 299–310 of: Pacific Symposium on Biocomputing.*

International human genome sequencing consortium, The. 2001. Initial Sequencing and analysis of the human genome. *Nature*, **409**, 860–921.

Jaakkola, T., Diekhans, M., & Haussler, D. 2000. A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, **7(1,2)**, 95–114.

Jain, A. K., & R. C. Dubes, R. C. 1988. *Algorithms for Clustering Data.* Prentice Hall.

Jain, A. K., Murty, M. N., & Flynn, P. J. 1999. Data clustering: a review. *ACM Computing Surveys*, **31**(3), 264–323.

Jaroszewicz, S., & Simovici, D. A. 2001. A General Measure of Rule Interestingness. *Pages 253–265 of: Proceedings of the 5th European Conference on Principles and Practice of Knowledge Discovery in Databases.*

Karalič, Aram, & Pirnat, Vlado. 1991. Significance Level Based Classification With Multiple Trees. *Informatica*, **15(5)**.

Karwath, A. 2002. *Large Logical Databases and their Applications to Computational Biology.* Ph.D. thesis, Department of Computer Science, University of Wales, Aberystwyth.

Karwath, A., & King, R.D. 2002. Homology Induction: The use of machine learning to improve sequence similarity searches. *BMC Bioinformatics 2002,*, **3:11**.

Kaufman, K. A., & Michalski, R. S. 1996. A Method for Reasoning with Structured and Continuous Attributes in the INLEN-2 Multistrategy Knowledge Discovery System. *Pages 232–237 of: Knowledge Discovery and Data Mining.*

Kell, D., & King, R. 2000. On the optimization of classes for the assignment of unidentified reading frames in functional genomics programmes: the need for machine learning. *Trends Biotechnol.*, **18**(March), 93–98.

Kennedy, C., & Giraud-Carrier, C. 1999 (April). An Evolutionary Approach to Concept Learning with Structured Data. *In: 4th International Conference on Artificial Neural Networks and Genetic Algorithms.*

Kennedy, C., Giraud-Carrier, C., & Bristol, D. 1999 (Sep). Predicting Chemical Carcinogenesis using Structural Information Only. *In: 3rd European Conference on the Principles of Data Mining and Knowledge Discovery.*

Khodursky, A., Peter, B., Cozzarelli, N., Botstein, D., & Brown, P. 2000. DNA microarray analysis of gene expression in response to physiological and genetic changes that affect tryptophan metabolism in *Escherichia coli*. *Proc. Nat. Acad. Sci. USA*, **97(22)**(Oct), 12170–12175.

King, R., & Srinivasan, A. 1996. Prediction of Rodent Carcinogenicity Bioassays from Molecular Structure Using Inductive Logic Programming. *Environmental Health Perspectives*, **104(5)**, 1031–1040.

King, R., Muggleton, S., Srinivasan, A., & Sternberg, M. 1996. Structure-activity relationships derived by machine learning: The use of atoms and their bond connectives to predict mutagenicity by inductive logic programming. *Proc. Nat. Acad. Sci. USA*, **93**(Jan), 438–442.

King, R., Karwath, A., Clare, A., & Dehaspe, L. 2000a. Accurate prediction of protein functional class in the *M. tuberculosis* and *E. coli* genomes using data mining. *Comparative and Functional Genomics*, **17**, 283–293.

King, R., Karwath, A., Clare, A., & Dehaspe, L. 2000b. Genome Scale Prediction of Protein Functional Class from Sequence Using Data Mining. *In: KDD 2000.*

King, R., Karwath, A., Clare, A., & Dehaspe, L. 2001. The Utility of Different Representations of Protein Sequence for Predicting Functional Class. *Bioinformatics*, **17(5)**, 445–454.

King, R. D., Garrett, S., & Coghill, G. 2000c. Bioinformatic System Identification. *In: Proc. 2nd Int. Conf. on Bioinformatics of Genome Regulation and Structure, Novosibirsk, Russia.*

Klein, P., Kanehisa, M., & DeLisi, C. 1985. The detection and classification of membrane-spanning proteins. *Biochim. Biophys. Acta*, **815**, 468–476.

Kohavi, R. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. *In: IJCAI 1995.*

Kohavi, R., & Sahami, M. 1996. Error-Based and Entropy-Based Discretization of Continuous Features. *Pages 114–119 of: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining.*

Kohavi, Ron, Brodley, Carla, Frasca, Brian, Mason, Llew, & Zheng, Zijian. 2000. KDD-Cup 2000 Organizers' Report: Peeling the Onion. *SIGKDD Explorations*, **2**(2), 86–98.

Koller, D., & Sahami, M. 1997. Hierarchically classifying documents using very few words. *In: ICML 97*.

Komorowski, J., & Øhrn, A. 1999. Diagnosing Acute Appendicitis with Very Simple Classification Rules. *Page 462467 of: Proc. Third European Symposium on Principles and Practice of Knowledge Discovery in Databases*.

Koonin, E., Tatusov, R., Galperin, M., & Rozanov, M. 1998. Genome analysis using clusters of orthologous groups (COGS). *Pages 135–139 of: RECOMB 98*.

Kowalczuk, M., Mackiewicz, P., Gierlik, A., Dudek, M., & Cebrat, S. 1999. Total Number of Coding Open Reading Frames in the Yeast Genome. *Yeast*, **15**, 1031–1034. Also see http://smorfland.microb.uni.wroc.pl/numORFs.htm.

Koza, J.R., Mydlowec, W., Lanza, G., Yu, J., , & Keane, M.A. 2001. Reverse Engineering of Metabolic Pathways from Observed Data Using Genetic Programming. *Pages 434–445 of: Pacific Symposium on Biocomputing*.

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

Kuan, J., & Saier Jr., M.H. 1993. The mitochondrial carrier family of transport proteins: structural, functional, and evolutionary relationships. *Crit. Rev. Biochem. Mol. Biol.*, **28(3)**, 209–33.

Kumar, A., Cheung, K.-H., Ross-Macdonald, P., Coelho, P.S.R., Miller, P., & Snyder, M. 2000. TRIPLES: a Database of Gene Function in S. cerevisiae. *Nucleic Acids Res.*, **28**, 81–84.

Kurhekar, M.P., Adak, S., Jhunjhunwala, S., & Raghupathy, K. 2002. Genome-Wide Pathway Analysis and Visualization Using Gene Expression Data. *Pages 462–473 of: Pacific Symposium on Biocomputing*.

Langley, P. 1998. The Computer-Aided Discovery of Scientific Knowledge. *Pages 25–39 of: Proceedings of the First International Conference on Discovery Science*.

Lavrač, N., Flach, P., & Zupan, B. 1999. Rule Evaluation Measures: A Unifying View. *Pages 174–185 of: Ninth International Workshop on Inductive Logic Programming (ILP'99)*.

Lee, S. D., & De Raedt, L. 2002. Constraint based mining of first order sequences in SeqLog. *In: First International Workshop on Knowledge Discovery in Inductive Databases (KDID'02).*

Leroy, G., & Chen, H. 2002. Filling Preposition-Based Templates to Capture Information from Medical Abstracts. *Pages 350–361 of: Pacific Symposium on Biocomputing.*

Li, W. 1999. Statistical properties of open reading frames in complete genome sequences. *Computers and Chemistry*, **23**, 283–301.

Liang, S., Fuhrman, S., & Somogyi, R. 1998. REVEAL, A General Reverse Engineering Algorithm for Inference of GeneticNetwork Architectures. *Pages 18–29 of: Pacific Symposium on Biocomputing.*

Liu, B., Hsu, W., & Ma, Y. 1999. Mining Association Rules with Multiple Minimum Supports. *Pages 337–341 of: Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*

Liu, Y., Engelman, D. M., & Gerstein, M. 2002. Genomic analysis of membrane protein families: abundance and conserved motifs. *Genome Biology*, **3(10)**.

Loewenstern, D., Hirsch, H., Yianilos, P., & Noordewier, M. 1995 (Apr). *DNA Sequence Classification Using Compression-Based Induction.* Tech. rept. 95-04. DIMACS.

Lorenzo, D. 1996. Application of Clausal Discovery to Temporal Databases. *Pages 25–40 of: Proceedings of the MLnet Familiarization Workshop on Data Mining with Inductive Logic Programing.*

Lukashin, A., & Fuchs, R. 2001. Analysis of temporal gene expression profiles: clustering by simulated annealing and determining the optimal number of clusters. *Bioinformatics*, **17(5)**(May), 405–414.

Lussier, M., White, A., Sheraton, J., di Paolo, T., Treadwell, J., Southard, S., Horenstein, C., Chen-Weiner, J., Ram, A., Kapteyn, J., Roemer, T., Vo, D., Bondoc, D., Hall, J., Zhong, W., Sdicu, A., Davies, J., Klis, F., Robbins, P., & Bussey, H. 1997. Large scale identification of genes involved in cell surface biosynthesis and architecture in *Saccharomyces cerevisiae*. *Genetics*, **147**(Oct), 435–450.

M. Steinbach, M., Karypis, G., & Kumar, V. 2000. A comparison of document clustering techniques. *In: KDD Workshop on Text Mining.*

Maggio, E.T., & Ramnarayan, R. 2001. Recent developments in computational proteomics. *Trends Biotechnol*, **19**, 266–272.

Marcotte, E., Pellegrini, M., Thompson, M., Yeates, T., & Eisenberg, D. 1999. A combined algorithm for genome-wide prediction of protein function. *Nature*, **402**(Nov), 83–86.

McCallum, A. 1999. Multi-Label text classification with a mixture model trained by EM. *In: AAAI 99 Workshop on Text Learning.*

McCallum, A., Rosenfeld, R., Mitchell, T., & Ng, A. 1998. Improving Text Classification by Shrinkage in a Hierarchy of Classes. *In: ICML 98.*

Mewes, H.W., Heumann, K., Kaps, A., Mayer, K., Pfeiffer, F., Stocker, S., & Frishman, D. 1999. MIPS: a database for protein sequences and complete genomes. *Nucleic Acids Research*, **27**, 44–48.

MGED Working Group, The. 2001. *Minimum Information About a Microarray Experiment.* http://www.mged.org/Annotations-wg/index.html.

Michalski, R. 1969. On the quasi-minimal solution of the covering problem. *Pages 125–128 of: Proceedings of the 5th International Symposium on Information Processing (FCIP-69), volume A3 (Switching Circuits).*

Michie, D. 1986. The superarticulacy phenomenon in the context of software manufacture. *Proceedings of the Royal Society of London, A*, **405**, 185–212.

Minsky, M. L., & Papert, S. A. 1969. *Perceptrons.* MIT Press.

Mitchell, T. 1997. *Machine Learning.* McGraw-Hill, Singapore.

Mitchell, T. 1998 (Feb). *Conditions for the Equivalence of Hierarchical and Non-Hierarchical Bayesian Classifiers.* Technical Note. Carnegie Mellon University.

Mladenic, D., & Grobelnik, M. 1998. Learning document classification from large text hierarchy. *In: AAAI 98.*

Möller, S., Leser, U., Fleischmann, W., & Apweiler, R. 1999. EDITtoTrEMBL: a distributed approach to high-quality automated protein sequence annotation. *Bioinformatics*, **15(3)**, 219–227.

Morik, K. 2000. The representation race: Preprocessing for handling time phenomena. *Pages 4–19 of: ECML 2000.*

Mueller, A. 1995 (August). *Fast sequential and parallel algorithms for association rule mining: A comparison.* Tech. rept. CS-TR-3515. Department of Computer Science, University of Maryland.

Muggleton, S. 1995. Inverse Entailment and Progol. *New Gen. Comput.*, **13**, 245–286.

Muggleton, S. 1996. Learning from positive data. *Pages 358–376 of: Proceedings of the 6th International Workshop on Inductive Logic Programming, volume 1314 of Lecture Notes in Artifcial Intelligence.*

Muggleton, S., & Feng, C. 1990. Efficient induction of logic programs. *Pages 368–381 of: Proceedings of the 1st Conference on Algorithmic Learning Theory.* Ohmsma, Tokyo, Japan.

Muggleton, S., King, R., & Sternberg, M. J. E. 1992. Protein secondary structure prediction using logic-based machine learning. *Protein Engineering*, **5(7)**, 647–657.

Myers, E., *et al.*. 2000. A Whole-Genome Assembly of Drosophila. *Science*, **287**, 2196–2204.

Newton, M.A., Kendziorski, C.M., C.S., Richmond, Blattner, F.R., & Tsui, K.W. 2001. On differential variability of expression ratios: Improving statistical inference about gene expression changes from microarray data. *Journal of Computational Biology*, **8**, 37–52.

Oliver, S. 1996. A network approach to the systematic analysis of yeast gene function. *Trends in Genetics*, **12(7)**, 241–242.

Oliver, S., Winson, M., Kell, D., & Baganz, F. 1998. Systematic functional analysis of the yeast genome. *Trends Biotechnol.*, **16**(September), 373–378.

Ouali, M., & King, R.D. 2000. Cascaded multiple classifiers for secondary structure prediction. *Protein Science*, **9(6)**(Jun), 1162–76.

Padmanabhan, B., & Tuzhilin, A. 1999. Unexpectedness as a measure of interestingness in knowledge discovery. *Decision Support Systems*, **27(3)**, 303–318.

Page, R.D.M., & Cotton, J.A. 2002. Vertebrate Phylogenomics: Reconciled Trees and Gene Duplications. *Pages 536–547 of: Pacific Symposium on Biocomputing.*

Park, J., Teichmann, S.A., Hubbard, T., & Chothia, C. 1997. Intermediate sequences increase the detection of homology between sequences. *J Mol Biol*, **273(1)**(Oct), 349–54.

Park, J. S., Chen, M., & Yu, P. 1995a. An effective hash-based algorithm for mining association rules. *Pages 175–186 of: Proceedings of the 1995 ACM-SIGMOD International Conference on Management of Data.*

Park, J. S., Chen, M., & Yu, P. 1995b. Efficient parallel data mining for assocation rules. *In: CIKM '95.*

Parthasrathy, S., Zaki, M., Ogihara, M., & Li, W. 2001. Parallel Data Mining for Association Rules on Shared-memory Systems. *Knowledge and Information Systems*, **3(1)**(Feb), 1–29.

Pawlowski, K., Jaroszewski, L., Rychlewski, L., & Godzik, A. 2000. Sensitive Sequence Comparison as Protein Function Predictor. *Pages 42–53 of: Pacific Symposium on Biocomputing.*

Pennisi, E. 1999. Keeping Genome Databases Clean and Up to Date. *Science*, **286**(Oct), 447–450.

Perneger, T. V. 1998. What's wrong with Bonferroni adjustments. *BMJ*, **316**, 1236–1238. See also the Responses to this article, including "What's wrong with arguments against multiplicity adjustments", Bender, R. and Lange, S.

Provost, F., & Kolluri, V. 1999. A survey of methods for scaling up inductive algorithms. *Data Mining and Knowledge Discovery*, **3(2)**, 131–169.

Provost, F., Fawcett, T., & Kohavi, R. 1998. The case against accuracy estimation for comparing induction algorithms. *Pages 445–453 of: Proc. 15th International Conf. on Machine Learning.* Morgan Kaufmann, San Francisco, CA.

Pustejovsky, J., J., Castaño., Zhang, J., Kotecki, M., , & Cochran, B. 2002. Robust Relational Parsing Over Biomedical Literature: Extracting Inhibit Relations. *Pages 362–373 of: Pacific Symposium on Biocomputing.*

Quinlan, J. R. 1993. *C4.5: programs for Machine Learning.* Morgan Kaufmann, San Mateo, California.

Raamsdonk, L. M., Teusink, B., Broadhurst, D., Zhang, N., Hayes, A., Walsh, M. C., Berden, J. A., Brindle, K. M., Kell, D. B., Rowland, J. J., Westerhoff, H. V., van Dam, K., & Oliver, S. G. 2001. A functional genomics strategy that uses metabolome data to reveal the phenotype of silent mutations. *Nature Biotech*, 45–50.

Ram, A., Wolters, A., Ten Hoopen, R., & Klis, F. 1994. A new approach for isolating cell wall mutants in *Saccharomyces cerevisiae* by screening for hypersensitivity to calcofluor white. *Yeast*, **10**, 1019–1030.

Rastan, S., & Beeley, L. 1997. Functional genomics: going forwards from the databases. *Current Opinion in Genetics and Development*, **7**, 777–783.

Reiser, P.G.K., King, R.D., Kell, D.B., Muggleton, S.H., Bryant, C.H., & Oliver, S.G. 2001. Developing a Logical Model of Yeast Metabolism. *Electronic Transactions in Artificial Intelligence*.

Richard, G., Fairhead, C., & Dujon, B. 1997. Complete transcriptional map of yeast chromosome XI in different life conditions. *Journal of Molecular Biology*, **268**, 303–321.

Riley, M. 1993. Functions of the gene products of *E. coli*. *Microbiol. Rev.*, **57**, 862–952.

Riley, M. 1998. Systems for categorizing functions of gene products. *Current Opinion in Structural Biology*, **8**, 388–392.

Rodríguez, J., Alonso, C., & Boström. 2000. Learning First Order Logic Time Series Classifiers. *In: Tenth International Conference on Inductive Logic Programming*.

Ross-Macdonald, P et al. 1999. Large-scale analysis of the yeast genome by transposon tagging and gene disruption. *Nature*, **402**(Nov), 413–418.

Rost, B., & Sander, C. 1993. Prediction of protein secondary structure at better than 70% accuracy. *Journal of Molecular Biology*, **232(2)**, 584–99.

Roth, F., Hughes, J., Estep, P., & Church, G. 1998. Fining DNA regulatory motifs within unaligned noncoding sequences clustered by whole-genome mRNA quantitation. *Nature Biotechnology*, **16**(October), 939–945.

Rubin, D. L., Shafa, F., Oliver, D. E., Hewett, M., , & Altman, R. B. 2002. Representing genetic sequence data for pharmacogenomics: an evolutionary approach using ontological and relational models. *Bioinformatics*, **18**, S207–S215.

Rumelhart, D. E., & McClelland, J. L. 1986. *Parallel Distributed Processing*. Vol. 1. MIT Press.

Sahar, S. 1999. Interestingness via What is Not Interesting. *Pages 332–336 of: Fifth International Conference on Knowledge Discovery and Data Mining*.

Salzberg, S. 1997. On Comparing Classifiers: Pitfalls to Avoid and a Recommended Approach. *Data Mining and Knowledge Discovery*, **1**, 317–327.

Salzberg, S., Chen, X., Henderson, J., & Fasman, K. 1996. Finding Genes in DNA using Decision Trees and Dynamic Programming. *Pages 201–210 of: ISMB '96*.

Sarasere, A., Omiecinsky, E., & Navathe, S. 1995. An efficient algorithm for mining association rules in large databases. *In: 21st International Conference on Very Large Databases (VLDB).*

Schapire, R., & Singer, Y. 2000. BoosTexter: A boosting-based system for text categorization. *Machine Learning*, **39(2/3)**, 135–168.

Schonbrun, J., Wedemeyer, W. J., & D., Baker. 2002. Protein structure prediction in 2002. *Curr Opinion Struct Biol.*, **12(3)**, 348–354.

Senes, A., Gerstein, M., & Engelman, D. M. 2000. Statistical analysis of amino acid patterns in transmembrane helices: The GxxxG motif occurs frequently and in association with β-branched residues at neighboring positions. *Journal of Molecular Biology*, **296**, 921–936.

Shah, I., & Hunter, L. 1997. Predicting Enzyme Function from Sequence: A Systematic Appraisal. *Pages 276–283 of: ISMB 97.*

Sharp, P.M., & WH Li, W.H. 1987. The Codon Adaptation Index–a measure of directional synonymous codon usage bias, and its potential applications. *Nucleic Acids Research*, **15**, 1281–1295.

Simon, I., Fiser, A., & Tusnady, G.E. 2001. Predicting protein conformation by statistical methods. *Biochim Biophys Acta*, **1549**, 123–136.

Spears, W. M., De Jong, K. A., Bäck, T., Fogel, D. B., & de Garis, H. 1993. An Overview of Evolutionary Computation. *Pages 442–459 of: Proceedings of the European Conference on Machine Learning (ECML-93)*, vol. 667.

Spellman, P, Sherlock, G., Zhang, M., Iyer, V., Anders, K., Eisen, M., Brown, P., Botstein, D., & Futcher, B. 1998. Comprehensive identification of cell cycle-regulated genes of the yeast *Saccharomyces cerevisiae* by microarray hybridization. *Molecular Biology of the Cell*, **9**(December), 3273–3297.

Srinivasan, A. 2001. Four Suggestions and a Rule Concerning the Application of ILP. *Pages 365–374 of:* Dzeroski, Saso, & Lavrac, Nada (eds), *Relational Data Mining.* Springer-Verlag.

Srinivasan, A., & Camacho, R.C. 1999. Numerical reasoning with an ILP program capable of lazy evaluation and customised search. *Journal of Logic Programming*, **40(2,3)**, 185–214.

Srinivasan, A., King, R. D., & Muggleton, S. 1999. *The role of background knowledge: using a problem from chemistry to examine the performance of an ILP program.* Tech. rept. PRG-TR-08-99. Oxford University Computing Laboratory, Oxford.

Sternberg, M. (ed). 1996. *"Protein Structure Prediction"*. IRL Press, Oxford.

Sugimoto, K., Sakamoto, Y., Takahashi, O., & Matsumoto, K. 1995. HYS2, an essential gene required for DNA replication in Saccharomyces cerevisiae. *Nucleic Acids Res*, **23(17)**(Sep), 3493–500.

Suzuki, E., Gotoh, M., & Y., Choki. 2001. Bloomy decision tree for multi-objective classification. *In: Proceedings of 5th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD 2001).*

Tatusov, R.L., Natale, D.A., Garkavtsev, I.V., Tatusova, T.A., Shankavaram, U.T., Rao, B.S., Kiryutin, B., Galperin, M.Y., Fedorova, N.D., & Koonin, E.V. 2001. The COG database: new developments in phylogenetic classification of proteins from complete genomes. *Nucleic Acids Res*, **29(1)**(Jan), 22–8.

Tavazoie, S., Hughes, J., Campbell, M., Cho, R., & Church, G. 1999. Systematic determination of genetic network architecture. *Nature Genetics*, **22**(July), 281–285.

Taylor, J., King, R.D., Altmann, Th., & Feihn, O. 2002. Application of metabolomics to plant genotype discrimination using statistics and machine learning. *Bioinformatics*, **18(S2)**, S241–S248.

Thomas, J., Milward, D., Ouzounis, C., Pulman, S., & Carroll, M. 2000. Automatic Extraction of Protein Interactions from Scientific Abstracts. *In: Pacific Symposium on Biocomputing 2000.*

Tomita, M. 2001. Whole cell simulation: A grand challenge of the 21st century. *Trends in Biotechnology*, **19:6**, 205–210.

Törönen, P., Kolehmainen, M., Wong, G., & Castrén, E. 1999. Analysis of gene expression data using self-organizing maps. *FEBS Lett.*, **451(2)**(May), 142–6.

Ullman, J. D. 1988. *Principles of Database and Knowledge-Base Systems, Vol. 1 and 2.* Computer Science Press, Rockville, Md.

Utgoff, P.E. 1986. Shift of bias for inductive concept learning. *In:* Michalski, R.S., Carbonell, J.G., & Mitchell, T.M. (eds), *Machine Learning: An Artificial Intelligence Approach, Volume II.* Morgan Kaufmann.

van Roermund, C. W. T., Drissen, R., van den Berg, M., Ijlst, L., Hettema, R. H., Tabak, H. F., Waterham, H. R., & Wanders, R. J. A. 2001. Identification of a Peroxisomal ATP Carrier Required for Medium-Chain Fatty Acid β-Oxidation and Normal Peroxisome Proliferation in *Saccharomyces cerevisiae*. *Molecular and Cellular Biology*, **21(13)**(July), 4321–4329.

Vapnik, V. 1998. *Statistical Learning Theory.* Wiley, NY.

Various. 1999. *The Chipping Forecast.* Nature Genetics, Volume 21, Supplement, January.

Velculescu, V., Zhang, L., Zhou, W., Vogelstein, J., Basrai, M., Bassett, D., Heiter, P., Vogelstein, B., & Kinzler, K. 1997. Characterization of the yeast transcriptome. *Cell*, **88**(Jan), 243–251.

Venter, J. C., *et al.*. 2001. The sequence of the human genome. *Science*, **291**, 1304–1351.

Vu, T. T., & Vohradsky, J. 2002. Genexp - a genetic network simulation environment. *Bioinformatics*, **18**, 1400–1401.

Walker, M., Volkmuth, W., Sprinzak, E., Hodgson, D., & Klingler, T. 1999. Prediction of gene function by genome-scale expression analysis: prostate cancer-associated genes. *Genome Res*, **9(12)**(Dec), 1198–1203.

Wang, K., Zhou, S., & He, Y. 2001. Hierarchical Classification of Real Life Documents. *In: Proceedings of the 1st SIAM International Conference on Data Mining.*

Wang, L.-S., Jansen, R.K., Moret, B.M.E., Raubeson, L.A., , & Warnow, T. 2002. Fast Phylogenetic Methods for the Analysis of Genome Rearrangement Data: An Empirical Study. *Pages 524–535 of: Pacific Symposium on Biocomputing.*

Waugh, A., Williams, G. A., Wei, L., & Altman, R. B. 2001. Using Metacomputing Tools to Facilitate Large Scale Analyses of Biological Databases. *Pages 360–371 of: Pacific Symposium on Biocomputing.*

Webb, E. (ed). 1992. *Enzyme Nomenclature 1992. Recommendations of the Nomenclature Committee of the International Union of Biochemistry and Molecular Biology.* Academic Press, San Diego, California.

Weber, Irene. 1998. A declarative language bias for levelwise search of first-order regularities. *Pages 106–113 of:* Wysotzki, Fritz, Geibel, Peter, & Schädler, Christina (eds), *Proc. Fachgruppentreffen Maschinelles Lernen (FGML-98).* TU Berlin.

Williams, H.E., & Zobel, J. 2002. Indexing and retrieval for genomic databases. *IEEE Transactions on Knowledge and Data Engineering*, **14(1)**, 63–78.

Wilson, C., Kreychman, J., & Gerstein, M. 2000. Assessing annotation transfer for genomics: Quantifying the relations between protein sequence, structure and function through traditional and probablaistic scores. *JMB*, **297**, 233–249.

Winzeler, E., & Davis, R. 1997. Functional analysis of the yeast genome. *Current Opinion in Genetics and Development*, **7**, 771–776.

Witten, I. H., & Frank, E. 1999. *Data Mining: Practical machine learning tools with Java implementations*. Morgan Kaufmann, San Francisco.

Yang, Y., & Pedersen, J. O. 1997. A comparative study on feature selection in text categorization. *Pages 412–420 of: Proceedings of ICML-97, 14th International Conference on Machine Learning*. Morgan Kaufmann Publishers, San Francisco, US.

Yeung, K.Y., Haynor, D., & Ruzzo, W. 2001. Validating clustering for gene expression data. *Bioinformatics*, **17(4)**, 309–318.

Yu, J., *et al.*. 2002. A draft sequence of the rice genome (*Oryza sativa* L. ssp. *indica*). *Science*, **296**, 79–92.

Zavaljevski, N., Stevens, F. J., & Reifman, J. 2002. Support vector machines with selective kernel scaling for protein classification and identification of key amino acid positions. *Bioinformatics*, **18**, 689–696.

Zhang, M. 1999. Large Scale Gene Expression Data Analysis: A New Challenge to Computational Biologists. *Genome Research*, **9**, 681–688.

Zhou, Y., Huang, G. M., & Wei, L. 2002. UniBLAST: a system to filter, cluster, and display BLAST results and assign unique gene annotation. *Bioinformatics*, **18**, 1268.

Index