Knowledge Discovery from cDNA Microarrays and *a priori* Knowledge

Herman Midelfart

04 Sep 2003

Department of Computer and Information Science
Norwegian University of Science and Technology
NO-7491 Trondheim, Norway

URN:NBN:no-7286

Knowledge Discovery from cDNA Microarrays and *a priori* Knowledge
Herman Midelfart

NTNU Trondheim
Norge teknisk-naturvitenskapelig universitet
Doktor ingeniøravhandling 2003:65
Institutt for datateknikk og informasjonsvitenskap
IDI-rapport 06/03

ISBN 82-471-5617-2
ISSN 1503-416X

Abstract

Microarray technology has recently attracted a lot of attention. This technology can measure the behavior (i.e., RNA abundance) of thousands of genes simultaneously, while previous methods have only allowed measurements of single genes. By enabling studies on a genome-wide scale, microarray technology is currently revolutionizing biological research and creating a wide range of research opportunities. However, the technology generates a vast amount of data that cannot be handled manually. Computational analysis is thus a prerequisite for the success of this technology, and research and development of computational tools for microarray analysis are of great importance.

This thesis develops supervised learning methods based on Rough Set Theory (RST) for analyzing microarray data together with prior knowledge. Two kinds of microarray studies are considered.

The first is cancer studies where supervised learning may be used for predicting tumor subtypes and clinical parameters. We introduce a general RST approach for classification of tumor samples analyzed by microarrays. This includes a feature selection method for selecting genes that discriminate significantly between a set of classes. RST classifiers are then learned from the selected genes.

The approach is applied to a data set of gastric tumors. Classifiers for six clinical parameters are developed and demonstrate that these parameters can be predicted from the expression profile of gastric tumors. Moreover, the performance of the feature selection method as well as several learning and discretization methods implemented in Rosetta are examined and compared to the performance of linear and quadratic discrimination analysis. The classifiers are also biologically validated. One of the best classifiers is selected for each clinical parameter, and the connection between the genes used in these classifiers and the parameters are compared to the established knowledge in the biomedical literature. Many of these genes have no previously known connection to gastric cancer and provide interesting targets for further biological research.

The second kind of study is prediction of gene function from expression profiles measured with microarrays. A serious problem in this case is that functional classes, which are assigned to genes, are typically organized in an ontology where the classes may be related to each other. One example is the Gene Ontology where the classes form a Directed Acyclic Graph (DAG). Standard learning methods such as RST assume, however, that the classes are unrelated, and cannot deal with this problem directly. This thesis gives a solution by introducing an extended RST framework and two novel algorithms for learning in a DAG.

The DAG also constitutes a problem when a classifier is to be evaluated since standard performance measures such as accuracy or AUC do not recognize the structure of the DAG. Therefore, several new performance measures are introduced.

The algorithms are first tested on a data set that was created from human fibroblast cells by the means of microarrays. They are then applied on artificial data in order to obtain a better understanding of their behavior, and their weaknesses and strengths are identified.

URN:NBN:no-7286

Acknowledgments

This thesis could not have been written without help and support of several people. I would like to acknowledge their efforts.

First of all, I would like to thank my supervisors Jan Komorowski and Astrid Lægreid for their advice, support, and collaboration. Jan has been my supervisor in computer science. He introduced me to rough set methods for analyzing data and awaken my interested in bioinformatics. Jan has also helped to open up several research opportunities for me for which I am grateful. Astrid has been my supervisor in biology and has helped me to acquire a better understanding of this field which I had only basic knowledge of before I began working on this thesis. In particular, she provided me with the opportunity of working with microarray technology and introduced me to the Gene Ontology. Astrid has been very helpful during all stages of this thesis and has always provided swift and constructive feedback upon my work.

In connection to this work, I have collaborated with several persons to whom I am grateful. These are, in alphabetical order, Torgeir R. Hvidsten, Kristin Nørsett, Arne K. Sandvik, and Fekadu Yadetie. Contact with Alte Bones and Per Winge also enhanced my understanding of biology and microarray technology. I have also benefited from discussions of my work with Torulf Mollestad.

I wish to thank my family for their support and encouragement during these years. In particular, I would like to thank my mother for all that she has done from me.

This work was sponsored mainly by the Department of Computer and Information Science at the Norwegian University of Science and Technology. I am grateful to Kjell Bratbergsengen and the Department of Computer and Information Science for extending my grant. It was much appreciated.

Herman Midelfart
August 2, 2003

Contents

List of Figures

List of Tables

List of Algorithms

Introduction

1

1.1 From data-poor to data-rich biology

Biological research is currently undergoing a revolution. Traditionally, biology has been a data-poor science. The previous methods for measuring biological phenomena have been limited and very time-consuming. Thus organisms have not been studied as complete entities. Rather, a reductionistic approach has been taken: An organism has been divided into increasingly smaller parts such as organs, cells, cell parts, and molecules (e.g., chromosomes). These parts have been then studied separately.

However, to understand how a cell functions it is necessary to understand how its parts, such as the genes, interact. This means the parts must be studied simultaneously under many different experimental conditions. Such studies will inevitably create large quantities of data. Hence, biological research requires tools to produce and analyze large data sets. Recently, such tools have begun to appear, and biology seems to be evolving into a data-rich science.

This development started with the genome projects whose mission was to find the DNA sequence of an organism and identify the genes within the sequence. These studies generated large amounts of data since the whole sequence was acquired – not only parts. For example, the human genome contains about 3 mega bases[1] [32]. However, DNA sequencing is not the only biotechnology that produces large amounts of data. Mass spectrometry allows a large number of proteins to be identified within a cell [186]. Microarray technology, which we will consider in this thesis, can measure the mRNA abundance of thousands of genes simultaneously. Hence, the trend towards increasing volumes of data is not limited to the genome projects.

A data-rich biology will not only be important for research in biology, but will also have an impact on research outside of biology. As biologists produce larger data sets, they will need new computational tools to handle the data, and this creates

[1]As is explained in detail in Chapter 2, a DNA sequence is constructed from 4 different bases. The length of a sequence is thus measured in bases. Hence, the human genome consists of 3 million bases.

new opportunities and challenges for research in computer science. For example, the genome projects generated large amounts of sequence data that needed to be stored and analyzed. For these purposes computers were essential, and these needs resulted in new computational problems such as DNA sequence analysis and development of DNA sequence databases. So from a computer science perspective, the development of a data-rich biology can be seen as a source of interesting research problems.

1.2 Functional genomics

The work of determining the DNA sequence has come to an end for many organisms. At the time of writing the sequence has been completed for 77 organisms [11] and even more are to come. However, the genome projects have only identified the genes. It is still unknown what many of the genes do and how they interact. The next step is to determine the function of the all the genes. This is the topic of the new field of functional genomics.

One of the first tasks of functional genomics is to annotate the genes in an organism. This means assigning some element of function to a gene [186], and this assignment should preferentially be done in an efficient and systematic fashion.

However, the ultimate goal is more ambitious. The biologists want to understand how the genes work together in functioning cells and organisms and do not want just a list of genes and their functions [99]. Rather they want a network or a map of how the genes interact and regulate each other. This may bring answers to questions such as what goes wrong in a disease, how we age, and what happens if we lose a gene function.

As many sequence projects have been completed, there is an increasing interest in functional genomics at the moment. There is really a shift in research focus taking place, which is similar to the change in focus from syntax to semantics for programming languages in the 1970s. Research until now has been on identifying the genes – the symbols in the language, but now we want to know what they mean – their semantics.

However, to understand the semantics we need to measure the behavior of the genes, since the answers cannot be found directly from the sequence. The gene products, the proteins, or the intermediate product, the RNA, must be measured. Again, large quantities of data must be acquired since all of the genes must be considered simultaneously in order to get a picture of what is going on. For this purpose, microarray technology will be an important instrument, and the data generated with this technology will require new computational tools. So again, new computational problems are introduced as a consequence of a development in biology.

1.3 Objectives

Microarray technology was introduced in the mid-1990s [149, 55] and is a method for assessing the concentration of mRNA from a high number of genes in one assay.

Hence, it can measure how much a gene is expressed, i.e., how active a gene is in a cell. It is not the first method for measuring RNA levels. However, previous methods allow only one gene to be measured at a time, while a microarray can determine the concentrations for thousands of genes in parallel.

Some microarray studies have already been performed. Still, how this data should be analyzed and interpreted is not clear. Some authors present only lists of genes and their expression levels [37], while others use clustering for grouping similarly expressed genes [169, 48] (A comprehensive overview of microarray analysis is given in Chapter 3).

It is not likely that one single analysis will be useful in all situations. The questions, which biologists want to answer, are too diverse. Rather, the method should be chosen depending on the questions.

This thesis treats questions such as: What is the function of a gene? and What kind of tumor has a cancer patient? These questions can both be addressed with supervised learning methods. In this regard, one may distinguish between two kinds of problems depending on the kinds of objects being classified.

- In the first problem, the genes are classified such that functional descriptions are assigned to them. The functional descriptions are in this case the classes. A microarray forms a conditional attribute upon which the classification is based.

- In the second problem, a sample is taken from a patient and examined with a microarray. The microarray is in this case the object being classified while the genes form the attributes.

As will be discussed more extensively later, none of these problems can be handled directly by a standard supervised algorithm. In the first case, the functional classes that may be assigned to genes are organized in an ontology such that the classes are related. A supervised learning algorithm, however, assumes that the classes are unrelated. A new kind of learning algorithm is therefore required. In the second case, the number of samples that may be collected is much lower than the number of genes. The most interesting genes must thus be selected before a learning algorithm is applied. The objectives of this thesis are consequently to develop or adapt methods for such problems, and evaluate them using experimental data.

1.4 Results

This thesis contains both applied and theoretical results. The results are numerous and can be classified along the following dimensions:

- Contributions to Rough Set Theory and Machine Learning:
  - Advances to Rough Set Theory are made such that the ontology may be described formally. New operators are developed and some properties of these are proven.

  - Two novel algorithms for learning in an ontology are developed:
    * One is a bottom-up method. This tries to learn rules that predict the most detailed classes possible without sacrificing prediction accuracy. It creates rules by traversing the ontology in a bottom-up fashion. It starts with the leaf classes and tries to learn rules that have a high accuracy. If it fails to find such rules for a particular class, it pushes the genes annotated to this class up to a more general class and tries again to learn rules for these genes when it visits this more general class.
    * The other can be described as an ensemble method. It learns rules for each class in the ontology using both the genes annotated to a class and the genes annotated to its subclasses. Classification of new genes is then done with a special voting system which attempts to balance the accuracy and the detail level of the predictions that are made by the rules.

    These methods are tested on several data sets:
    * A data set containing measurements from an experiment on human fibroblast cells.
    * Various artificial data sets.
  - Classifiers built for an ontology also require that the ontology is taken into consideration when they are evaluated. Methods for such evaluation are therefore introduced.
  - It is demonstrated that Rough Set methods combined with a gene selection method may be used for classifying tumors.

- Contributions to Cancer Research:
  - Diagnostic classifiers for gastric tumors are created from microarray data and clinical parameters. These predict the outcome of a clinical parameter from a tumor sample which has been examined with a microarray.
  - It is shown that multiple classifiers can be derived from the same microarray data set.
  - The diagnostic classifiers, which are derived from the microarray data and the clinical parameters, identify genes that may have an impact on the parameters. These genes may be interesting targets for further biological research.

1.5 Structure of the thesis

The intended reader of this thesis should have a background in computer science or bioinformatics. However, large parts of this thesis should be accessible to a biologist.

No previous knowledge of rough set theory is required, but basic knowledge of discrete mathematics and computer science is assumed.

The thesis contains 10 chapters, besides this introduction. These are as follows:

Chapter 2 introduces to molecular biology and microarray technology.

Chapter 3 concerns itself with microarray analysis. The process involved in this analysis is explained and an overview on topics such normalization, clustering, and supervised learning of microarrays is given.

Chapter 4 explains the basic concepts in rough set theory such as indiscernibility relations and approximations. Moreover, decision rules are introduced, and several methods for learning such rules are discussed. Concepts such as voting are also considered.

Chapter 5 presents a general approach where rough set based learning is combined with a feature selection method and used for classification of tumor samples. The approach is applied to a data set of gastric tumors, which have been analyzed by microarrays. The performance of several discretization and learning methods is examined under different feature selection settings and determine best methods. The classifiers obtained with these methods are then inspected, and the genes used in them are compared to the established knowledge in the biomedical literature.

Chapter 6 gives a brief introduction to ontologies in general and the Gene Ontology in particular. Its discusses several issues, which have to be taken into account when the Gene Ontology is used for learning. The chapter also explains that the Gene Ontology may be considered as a DAG.

Chapter 7 introduces an extended rough set framework that allows a formal description of the ontology. Several new approximations are introduced and their properties are investigated.

Chapter 8 starts with a discussion on how the DAG complicates the learning process and describes several problems. It then presents two different algorithms, which solve these problems.

Chapter 9 presents several measures that can be used for evaluating the performance of a DAG classifier.

Chapter 10 reports the experimental results, which were obtained with the two algorithms given in Chapter 8. The algorithms are first applied to a microarray data set created from human fibroblast cells. They are then tested on artificial data, and their advantages and disadvantages are examined.

Chapter 11 summaries the thesis and draws conclusions. This chapter also points out several directions for further work.

1.6 Other publications

While preparing this thesis, several papers (which are not presented here) have been written or co-written by the author of this thesis. These may be found in: [111], [112], and [90].

Gene Expression Assessment

2

2.1 Introduction

This thesis documents work in the borderland between computer science and molecular biology. In particular, it discusses and introduces methods for microarray analysis. A general understanding of molecular biology and microarray technology is therefore essential.

This chapter introduces the necessary biological background. It also explains and surveys microarray technology. However, it is not the intention to give a detailed exposition of molecular biology and the chemistry of the cell. Rather, a conceptual view will be presented in order to make microarray technology comprehensible to a non-biologist such as a computer scientist. Details and terminology will be introduced only when necessary.

2.2 DNA, RNA, and proteins

The inherited information in an organism is stored in the *chromosomes* in the nucleus of its cells. A chromosome consists of a single DNA (deoxyribonucleic acid) molecule which again is composed of two strands.

A DNA strand is made up of nucleotides, which are attached to each other just like links in a chain. A nucleotide consists of a phosphate, a pentose (a carbon sugar group), and a base (see Figure 2.1). The phosphate and the pentose hold the strand together, while the base contains the actual genetic information. Only four different bases may occur in a nucleotide: Adenine, Guanine, Cytosine, and Thymine. The bases are often abbreviated with the letters (A, G, C, and T), and we may think of the strand as a string, i.e., a sequence of letters, constructed for this alphabet. Hence, the genetic code is quaternary code.

The two strands in a DNA molecule are bound together by base pairs. Two bases at corresponding positions in the two strands may bind to each other through hydrogen

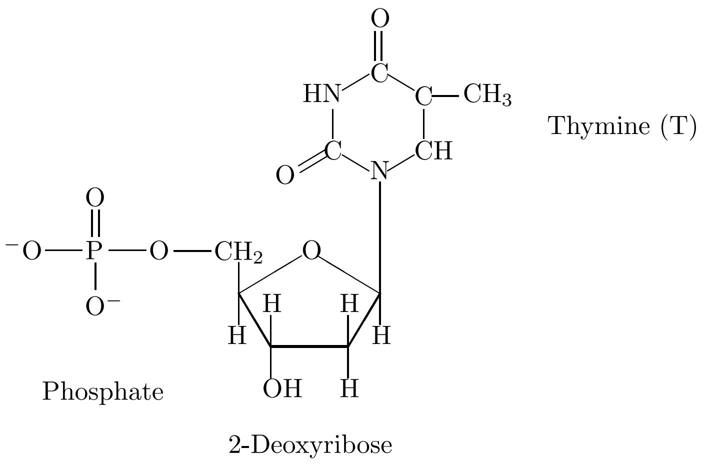


Figure 2.1: *A DNA nucleotide.* A nucleotide consists of a phosphate, a pentose, and a base (Thymine). The pentose shown here is 2-Deoxyribose and the nucleotide is a part of the a DNA molecule.

bonds and form a pair. Only two kinds of pairs occur ordinarily in the living cells: A-T and G-C. So if A occurs at a particular position in one strand, then T occurs at the corresponding position in the other. A is in this way a complement of T, and two strands (or regions of two strands) that obey this rule of pairing are said to be complementary, e.g. the following two strands are complementary:

```
... ATTTCG ...          Strand 1
    ||||||
... TAAAGC ...          Strand 2
```

Note that even though the DNA molecules in the nucleus have two strands, there are also DNA molecules with only one strand. It is customary to distinguish between these by referring to them as *double stranded* DNA and *single stranded DNA* (ssDNA). The bases in ssDNA do not form pairs since there is only one strand in the molecule. However, the base pairing mechanism allows ssDNA molecules to combine into a double stranded molecule. This will be discussed in Section 2.5.

An *RNA* (ribonucleic acid) molecule is a single stranded copy of (a region of) a DNA molecule except that the base Thymine is replaced by the base Uracil in an RNA sequence. The nucleotides in RNA are also slightly different from those in DNA. These nucleotides contain a pentose called ribose while DNA has a variant call 2-Deoxyribose.

RNAs can be divided into three categories: Messenger (mRNA), Transfer (tRNA), and Ribosomal (rRNA). mRNA contains genetic information that is used in the production of proteins (see below), while tRNA and rRNA participate in this production.

Proteins are involved in most of the activity in a cell. They participate in processes such as catalysis, defense, movement, protection, regulation, signaling, structural support, transport, transcription, and translation. The primary structure of a protein is a sequence made out of 20 different amino acids. As in the case of DNAs and RNAs, we may consider this structure as a string where the amino acids are abbreviated with either one-letter or three-letter codes. For example, the amino acid *Serine* is abbreviated either as *S* or *Ser*. The 20 one-letter-codes denoting amino acids in proteins should not be confused with the 5 letters denoting bases in DNA and RNA.

2.3 Information flow: From DNA to proteins

The *central dogma of molecular biology* states that the inherited information in the cell is processed in the following fashion:

$$\text{DNA} \rightarrow \text{RNA} \rightarrow \text{Protein}$$

RNAs are produced from DNAs, and proteins are created from RNAs. The production of RNAs is called *transcription*, while the synthesis of proteins is called *translation*.

Only parts of the DNA sequence encode RNAs. The rest of the DNA have other purposes. For example, centromers and teleomeres have special roles in DNA duplication. Still, there are regions whose purpose is unknown and may in fact have no function at all. The RNA encoding regions are called *genes* (see Figure 2.4), and a gene consists of a promoter region and a transcriptional unit. The promoter region controls the transcription, i.e., the production of RNA, of the gene, and the RNA is created from the transcriptional unit.

A gene is *expressed* in a cell if it is being transcribed. The transcription of a gene is activated by transcription factors that bind to binding sites in the promoter region of a gene. The transcription factors attract special enzymes called RNA polymerases which catalyze the synthesis of the RNA strands.

The RNA is transcribed from the one of the two strands in the DNA molecule by base pairing. The strand, which is used for transcription, is called the *template* strand, while the other is called the *complementary* strand. The synthesized RNA is complementary to the template due to the base pairing and hence identical to the complementary strand (except that Thymine is replaced by Uracil).

After the RNA has been transcribed, it undergoes some modifications, and only parts of the of the original transcriptional unit end up in the final mature RNA (see Figure 2.4). The transcriptional unit can be divided into regions called *exons* and *introns*. The exons are included in the mature RNA, and the introns are discarded. This is done in by cleavage and splicing. The RNA strand is cut at the start and end of an intron, and the ends of the neighboring exons are joined.

mRNA can then be translated into proteins. This is done by gathering the bases in the mRNA sequence into groups of 3. Such a group is called a *codon*, and each codon identifies one of the 20 amino acids that can occur in the protein sequence. Since a codon has a length of 3, and 4 bases may occupy each position, there are a

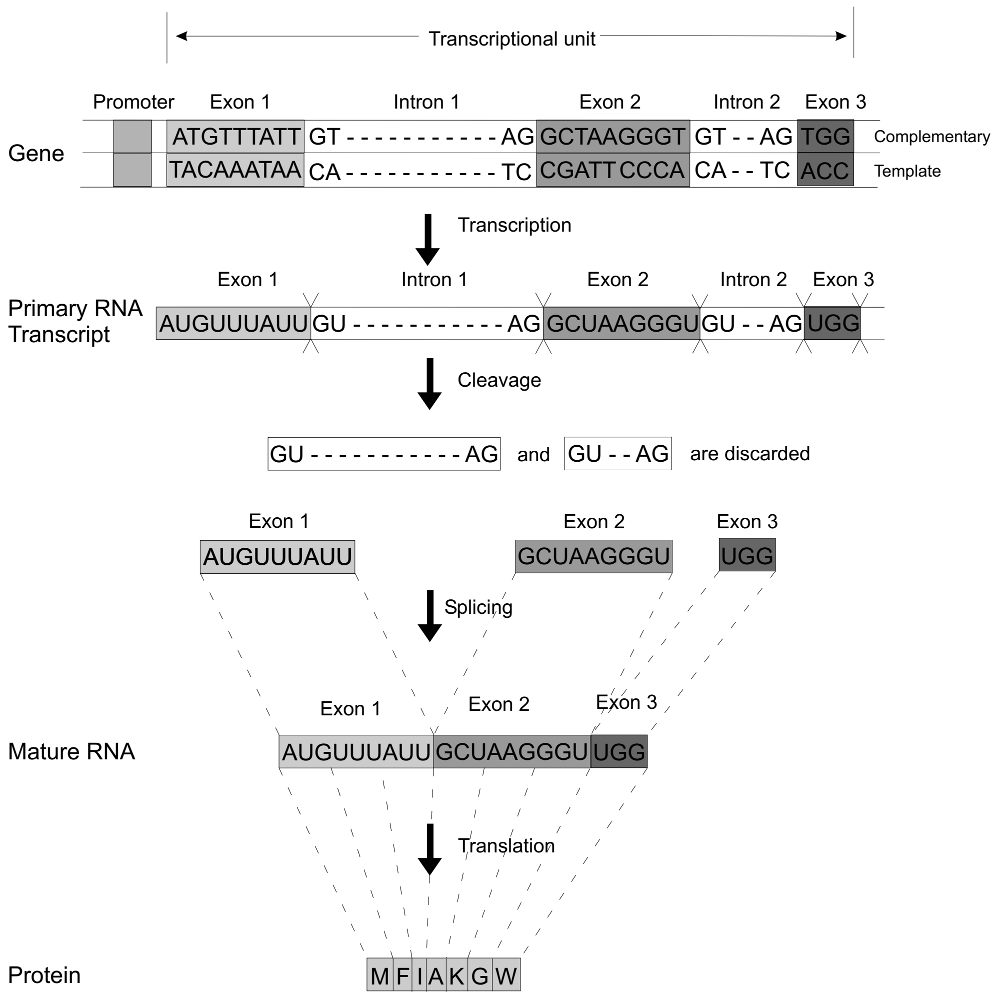

Transcriptional unit
Promoter
Exon 1
Intron 1
Exon 2
Intron 2
Exon 3
Gene
ATGTTTATT GT - - - - - - - - - - - AG GCTAAGGGT GT - - AG TGG
TACAAATAA CA - - - - - - - - - - - TC CGATT CCCA CA - - TC ACC
Complementary
Template
Transcription
Primary RNA Transcript
AUGUUUAUU GU - - - - - - - - - - - AG GCUAAGGGU GU - - AG UGG
Cleavage
GU - - - - - - - - - - - AG and GU - - AG are discarded
AUGUUUAUU
GCUAAGGGU
UGG
Splicing
Mature RNA
AUGUUUAUU GCUAAGGGU UGG
Translation
Protein
M F I A K G W

Figure 2.2: *Transcription and translation.* The DNA sequence of a gene is shown at the top. The primary RNA, which corresponds to the transcriptional unit is copied off the template strand and is later cleaved and spliced. This results in the mature RNA, which contains only the exons. The mature DNA is translated into a protein where each amino acid corresponds to a codon in the RNA.

total of $4^3 = 64$ different codons. This means that the genetic code is degenerative. Several codons may identify the same amino acid.

The translation is performed in a ribosome, which consists of rRNA and proteins. The ribosome moves along the mRNA strand and produces the protein strand. The actual matching of codons with amino acids is done by tRNAs. These molecules have a special region called an *anticodon* that recognizes codons on the mRNA strand and a site where one amino acid may attach itself. When a tRNA has bound to an amino acid, it may move into the ribosome and match a codon. If a codon is recognized, the amino acid is released and tRNA moves out of the ribosome. The released amino acid is attached to the protein strand being built by the ribosome.

A tRNA binds only to one kind of amino acid. Hence, different tRNAs are used for matching each of the 20 amino acids. The anticodon of a tRNA may still recognize several dissimilar codons coding for the same amino acid. The codons matched by the same anticodon may for example differ in the base at only one position. However, the anticodon will not match codons coding for other amino acids.

2.4 Reverse information flow

Though the inherited information flows from DNA to proteins with mRNA as an intermediate stage, there are mechanisms that work in the opposite direction. Proteins are involved in transcription and translation. In particular, transcription factors that activate transcription of genes are proteins. A gene may therefore regulate the activity of other genes indirectly through its protein(s). Hence, there is a loop back from proteins to DNA in the information flow.

A more direct reversal of the flow of information is *reverse transcription.* From a computer scientific perspective, one may compare DNA to the source code of a program where the code is stored in a non-volatile memory. The mRNA is a copy of the source code stored in a volatile memory, and proteins are processes that are running the compiled object code. Like source code, chromosomal DNA can be modified. An example of this can be found in the life cycle of retroviruses such as HIV where DNA is synthesized from RNA by reverse transcription. These viruses have only RNA as genome, and no DNA. When a retrovirus attacks a cell, it creates a DNA copy which is inserted into the host DNA. This tricks the host cell into producing RNA copies of the virus DNA and in this way reproducing the virus.

Reverse transcription is also used in the laboratories to create DNA from RNA. The process creates a single strand DNA that is complementary to RNA and therefore called *complementary DNA (cDNA).*

2.5 Nucleic acid hybridization

DNA is usually double stranded. However, the two strands may be separated experimentally into single stranded DNA by heating the DNA. This process of separation is called *denaturation*. If the temperature is lowered again the ssDNA will recombine to form double stranded DNA. Such regrouping of ssDNA is referred to as *hybridization*. However, only complementary ssDNA will hybridize and create double stranded DNA. Non-complementary strands will not form stable molecules. Hence, the strands in the recombined DNA molecules will match each other. Notice that hybridization is essentially performing a string matching operation of complementary strings. It could be compared to a parallel lexical analyzer.

This ability of matching strands makes hybridization an important tool for detecting the presence of a DNA sequence. By using a collection of known and identical ssDNA strands as a *probe* one can identify the complementary ssDNA strands in a *target* solution. The complementary target strands will bind to the probe strands when the probe is applied to the solution. The presence and abundance of the complementary strands can only be determined if the hybridized strands can be distinguished from the rest. This is done by labeling. Either the probe or target strands are marked with radioactive or fluorescent labels. The labeling consists of replacing the nucleotides in the strand with nucleotides having radioactive isotopes or fluorphores or adding such molecules to one of the ends of the strands. The radioactively labeled strands can be identified with X-ray film and the fluorescently labeled strands can be detected with a laser scanner.

Although only the recombination of ssDNAs has been mentioned so far, hybridization is not restricted to such molecules. Two RNA strands are also capable of such regrouping. Hybridization may also occur between an ssDNA strand and an RNA strand. However, RNAs are not very stable outside the cell and degrade quickly [140, p. 319]. They are, for this reason, often reverse transcribed into single stranded cDNAs when microarray technology is used. Hybridization takes then place between DNA strands.

2.6 DNA Cloning

The DNA in the nucleus is duplicated when a cell divides into two new cells. This action is catalyzed by DNA polymerases, which are similar to RNA polymerases, but produce DNA strands rather than RNA strands. A DNA polymerase copies only a single strand. Both strands of a double stranded DNA are copied simultaneously by separate enzymes starting from one end of the molecule and moving to the other end. The duplication is a bit more complicated since DNA polymerase can only produce copies in one direction. Complementary strands are oriented in opposite directions such that one strand is oriented in the wrong direction with regard to the duplication process. However, these details will not be considered here. The duplicated DNA is called a *clone*.

This replication process is used in laboratories to multiply a DNA strand for experimental purposes. In this case, the DNA is inserted into an organism such as the bacterium *E.coli* where it is duplicated along with the DNA of the organism through cell division. The cloned DNA can then be extracted from the cell after a series of cell divisions.

DNA strands may alternatively be replicated by PCR (Polymerase Chain Reaction). In this case, the DNA is replicated outside a living organism. The process goes through a series of cycles where the DNA is denaturated and copied by DNA polymerases.

2.7 Microarray technology

Even though the genes in many organisms have been identified, their functions and interactions are unknown. Studies of RNA and protein levels in a cell may reveal this. However, methods for measuring RNA (or protein) levels on a large scale have not been available until recently.

The traditional methods for measuring gene expression such as Dot-blot and Northern blot can only measure one gene at a time. In these methods a denatured target solution is fixed to a membrane and a labeled probe for one particular gene is applied to the membrane such that the probe and the target may hybridize. However, most organisms have thousands of genes. The human genome consists of $30,000 - 40,000$ protein coding genes [32] and yeast which has one of the smaller genomes among the eukaryotic organisms has $6,000$ genes [110]. Hence, only very limited studies of gene expression have been possible with these methods. Microarray technology on the other hand allows the expression level of $5,000$ up to $40,000$ genes to be measured simultaneously one array. In this way, it represents a breakthrough for genome studies.

A microarray is a glass slide or wafer. Probes for different genes are placed at fixed locations on the glass. Each probe appears as a circular or rectangular spot on the microarray. The location of a spot identifies the probe and therefore also the gene that matches the probe. The same probe can be placed at several locations so that a gene may be examined at several different spots. A gene may also be represented with different parts of its sequence such that two spots may inspect the same gene even though the probes at these locations may have different sequences.

One or two target samples are examined by a microarray. These samples are first labeled with fluorescent dyes and then applied to the glass plate where the DNA in samples hybridize with the probes on the plate. Excess target DNA is then washed off, and the abundance of labeled DNA in each spot is determined with a laser scanner.

There are several different microarray technologies. Their main distinguishing features are the production method, the probes, and the experimental design. These features depend often on each other. One particular probe may, for example, be used with one particular production method. A summary of the features is presented in Table 2.1.

Microarrays have also another application beside expression studies. They can be used for detecting SNPs (Single Nucleotide Polymorphisms) in the genome. These are variations found in individuals and result in different variants of the same gene. A survey of microarray methods for this application can be found in [66].

Features	***Options***	***Spotted microarray***	***Affymetrix' GeneChip***
Production	1. Spotting 2. Photolithography 3. Ink-jet	Spotting	Photo-lightography
Probes	1. Oligonucleotides 2. cDNAs	cDNAs or Oligonucleo-tides	Oligonucleo-tides
Experimental Design	1. One sample with PM and MM probes 2. Relative comparison be-tween two samples	Option 2	Option 1

Table 2.1: *Key features of microarray technology.*

2.7.1 Probes

cDNAs and oligonucleotides[1] are both used as probes on microarrays. cDNAs may have a length of 100 to 2000 base pairs, and oligonucleotides are typically 15-100 bases long. They have several distinguishing properties as probes where oligonucleotide probes have more advantages than cDNAs. This is mostly due to some properties of hybridization. Hybridization is not a well-understood process [100, p. 224], but it is known that the following factors affect the formation of double stranded molecules.

- **Sequence length**: A long sequence is in principle more specific and less likely to occur in several genes. Hence, the probability of hybridization between a probe and a mismatching target strand decreases with length when only the length is considered. However, the matching of strands is not necessary perfect, and mismatches may occur. The ability to form stable molecules with mismatches increases with length. A probe with a short sequence will not form stable double stranded molecules with target strands if the sequences differ at a single position, while a probe with a long sequence will (see Figure 2.3). The location of the mismatched base in the sequence has an affect as well. A mismatch in the center of the sequence is much more destabilizing than a mismatch near an end [167].

- **Sequence composition**: The complementary pairs A-T and G-C do not have the same strength of attraction. Cytosine and Guanine bind to each other through 3 hydrogen bonds while Adenine and Thymine have only 2 hydrogen bonds. Sequences with a high frequencies of Cs and Gs will attract each other more strongly and form more stable duplexes than sequences with a high frequencies of As and Ts [167].

[1]Oligonucleotides are short (typically 15-100 bases long) chemically synthesized DNA strands.

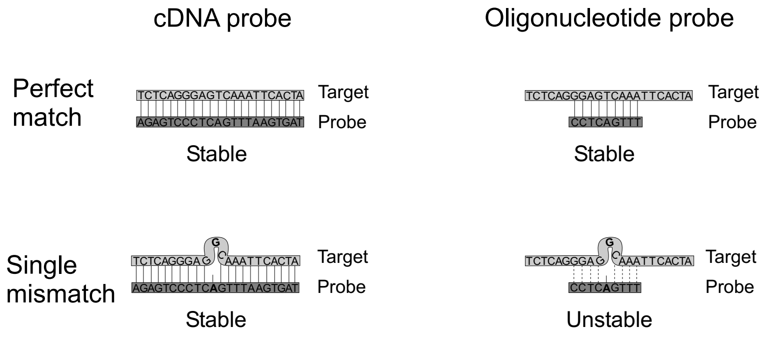


Figure 2.3: *Matching of probes.* The figure illustrates matching of a probe and a target strand when a long cDNA probe and a short oligonucleotide probe are used. In the first row, the target strand matches the cDNA probe and the oligonucleotide probe perfectly. Stable molecules are thus formed in both cases. In the second row, the middle base of target strand is not matched. In this case, the cDNA probe may form a stable molecule, while the oligonucleotide probe will not. The oligonucleotide probe is thus able to detect this discrepancy.

Hence, the specificity of a probe does not solely depend on the length. A short oligonucleotide may discriminate more effectively against non-matching strands than a long cDNA strand [74]. Cross-hybridization, i.e., binding of targets for several different genes to one probe on the array, is less likely. Discrimination of very similar genes from a gene family can only be done with oligonucleotides. Moreover, short oligonucleotides can detect a difference in a single base while cDNAs cannot [167]. Oligonucleotide microarrays can be designed to both identify unknown SNPs and for studying the correlation of SNP with characteristics in individuals [95]. An oligonucleotide array can also be stored for a longer period of time [107].

Still, there are some advantages of using cDNA. They are cheaper and less susceptible to the difference in the stability of A-T and G-C pairs. A short oligonucleotide is more likely to have uneven amounts of these pairs. Hence, the oligonucleotides on an array may bind to the target with different degrees of efficiency resulting in unequal sensitivity.

2.7.2 Production technology

Mechanical microspotting

The most widely available technology was developed by Schena et al. [149]. This is mainly due to the cost. Both the production equipment and the production of microarrays are less expensive compared other methods [108, 21]. Microarrays are produced with this technology by depositing cloned cDNA onto a glass slide. Due to the use of cDNA as probes, it is usually referred to as cDNA microarray technology.

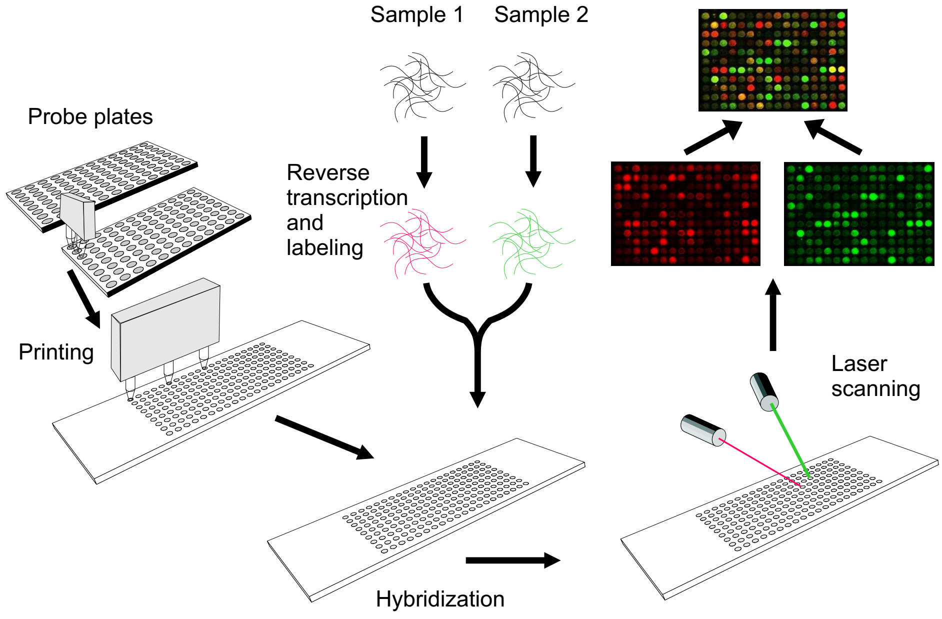


Figure 2.4: *Overview of cDNA microarray printing and scanning.* A robot deposits probes from plates onto a glass slide. Two samples are reverse transcribed and labeled with different fluorescent dyes. The samples are hybridized on the microarray which is scanned with a laser. This results in two images – one for each dye – which is combined into one image by computing the ratio between the spots.

The technology is flexible. The clones printed on the slide can be changed easily so that different arrays can be produced. It is also quite fast. 150 identical arrays of 12000 genes can be produced in a day [21]. One of the drawbacks is that a collection of cDNA has to be acquired and prepared and cataloged in advance of printing, which is time consuming [98]. Duggan et al. [40] have written a good review paper on this technology.

Figure 2.4 illustrates the production and scanning of cDNA microarrays. Probes are first printed by a robot onto a glass slide. The robot is equipped with a set of pins (also known as print tips) that are essentially capillary tubes. Each pin sucks up some cDNA from wells on probe plates, which are prepared in advance. The robot moves the pins to a fixed location on the glass slide and cDNA is deposited. This procedure is repeated until all the probes have been deposited.

Target RNA strands from two different samples – often a test and a reference sample – are reverse transcribed and labeled with two different fluorescent dyes Cy5 (Red) and Cy3 (Green). A mix of the labeled cDNA samples is applied to the microarray, and the target strands are permitted to bind to the complementary probe strands.

The remaining target strands that do not bind to any spot are then washed off. The microarray is scanned with a confocal laser using two different wavelengths. This results in two images – one for each color – with the intensity of each spot. The spot intensity is proportional to the concentration of the RNA in the sample and therefore a measure of concentration. For each spot a Cy5/Cy3-ratio is computed and corrected for background noise as follows:

$$Ratio = \frac{\text{Intensity of Cy5} - \text{Background intensity of Cy5}}{\text{Intensity of Cy3} - \text{Background intensity of Cy3}} \tag{2.1}$$

The background intensity is measured in the vicinity of the spot, and gives an assessment of the noise due to factors such as light reflections from the glass side, and fluorescence from labeled target cDNA, which has bound to the glass side without a matching probe [6, p. 277].

Even though cDNAs have mostly been used on spotted array, this technology is not limited to cDNAs. Oligonucleotides synthesized in advance can also be spotted on an array. A very early attempt was made by Guo et al. [61]. Kane et al. [80] studied the sensitive and the specificity of spotted oligonucleotides.

Photolithography

The main competitor to the spotted cDNA microarray is the so-called *GeneChip* manufactured by Affymetrix. The Affymetrix technology applies photolithography used in the production of semiconductors to perform a light-directed synthesis of oligonucleotides on a glass substrate. It was developed by Foder et al. [56, 55], and a recent overview is given in [95].

The synthesis of Affymetrix' microarray is explained in Figure 2.5. Chemical linkers are initially attached to a glass substrate. These serve as starting points for building the oligonucleotide strands, and are protected by photochemically removable groups. A nucleotide will only bind to a linker if the protection group is removed in advance. The oligonucleotides are produced by adding one nucleotide at time to the strands. Light is directed through a mask and the illuminated protection groups are released. Nucleotides with one particular base are added to surface and bind to the free linkers. These nucleotides have also photochemically protection groups such that no further binding is possible without additional lighting. The process of illumination and addition of nucleotides are repeated for the other bases using a different mask each time. A total of $4 \times N$ steps is necessary to produce all possible oligonucleotides with length N.

Only one sample is measured on a GeneChip. This sample is amplified in order to increase the sensitivity, and cRNA[2] is hybridized to the array instead of cDNA [95, 67]. Thus, the RNA in a sample is first reverse transcribed into cDNA. The cDNA is then transcribed in the presence of labeled nucleotides. The result is labeled cRNA, which is applied to the microarray. The microarray is later scanned with a

[2]cRNA is transcribed cDNA.

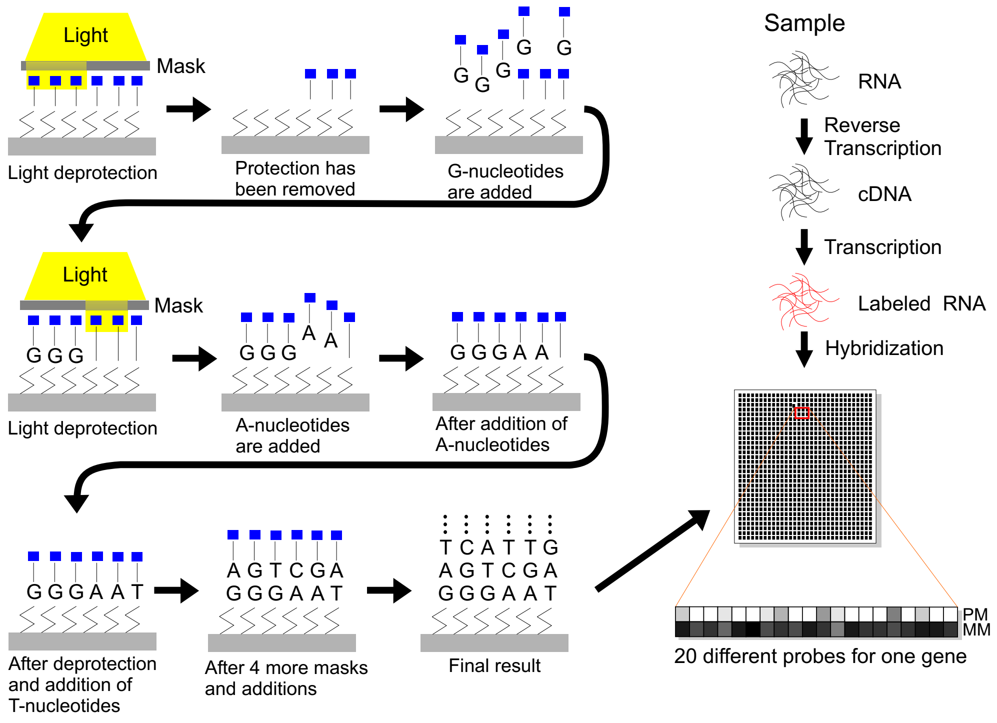


Figure 2.5: *Production of Affymetrix' GeneChip.* Oligonucleotides are synthesized in several iterations. In each iteration, protection groups are removed by illumination through a mask. Nucleotides with one particular base are then applied to the glass substrate such that they bind to the uncovered strands. The sample is amplified, fluorescently labeled, and hybridized on the array. The abundance of a gene is assessed with 20 different probe pairs where a pair examines a particular part of the sequence. Each pair consists of a perfect match probe (PM) and a mismatch probe (MM). The mismatch probe has a different base in the middle position.

confocal laser. Notice, however, that this amplification process may introduce errors since the efficiency of the transcription depends on the sequence composition. The relative abundance of the RNAs may consequently be changed.

The RNA concentration of each gene is assessed by 20 different probe pairs on a microarray [95, 67]. Each pair consists of a perfect match (PM) and a mismatch (MM) probe. A PM probe and its corresponding MM probe are identical except for the base in the middle of the sequence which is different. Each probe sequence has length of 25 nucleotides, and PM probes are taken from different regions of the gene sequence [95] such that they will hybridize to different parts of the RNA. The RNA abundance is calculated as average difference between PM and MM probes:

$$\frac{1}{20}\sum_{i=1}^{20}(PM_i - MM_i) \tag{2.2}$$

One of the advantages with this technology is that no clone collection is necessary. The probe strands are synthesized with information taken from a sequence database [151, 98]. The use of oligonucleotides as probes has also many benefits (as described in Section 2.7.1). However, the synthesis is quite slow. It takes about a day to synthesize an array with 25 nucleotide-long oligonucleotides [14]. A new mask is required for each nucleotide such that $4 \times N$ mask is needed for oligonucleotides with N nucleotides. The production of masks is expensive such that the technology is less flexible than microspotting [151].

Ink-jets

A third approach to the production of microarrays comes from ink-jet printers. Piezoelectric pumps used in the print head of such printers for applying ink to paper, are used to deposit fluids onto a glass slide. Fluids can be placed at exact locations by moving the print head over the array. This technology can be used for both microspotting of cDNAs or oligonucleotides [151, 145] and synthesis of oligonucleotides directly onto the glass [14, 74]. However, the synthesis is more desirable since no preparation of probes is required.

Ink-jet synthesis holds a great potential, and it may become the leading microarray technology. It is both faster and cheaper that Affymetrix' technology. Piezoelectric pumps can operate at very high frequencies and produce arrays much faster that photolithography. Blanchard et al. [14] suggest that an array of $100,000$ different oligonucleotides with a length of 25 nucleotides could be produced in less than 2 hours. No expensive masks are needed either. However, this technology is not completely mature and suffers from problems such as clogging of delivery heads and interruption of fluid supply by air bubbles [104].

2.7.3 Design

Two kinds of designs are applied on microarrays. The first type performs a relative comparison between two samples and is employed on spotted arrays. Usually, a single probe sequence is used for each gene. This design has also been used with ink-jet synthesized microarrays [74]. Often a test sample is compared to a reference sample. The reference sample may seem unnecessary in this case, but it is required since the printing technology is not perfect. The form and quality of the spots may vary from array to array. Two spots that are even printed in the same position on two arrays may be different such that the intensity is different. By measuring the abundance relative to a common reference sample and computing ratios, this error is removed since the error is mainly multiplicative.

Affymetrix' GeneChips is more uniform and finds the absolute abundance of the genes. It measures only one sample using several different short oligonucleotide probes for each gene. The use of several probes makes it possible to measure different regions of the sequence. This compensates for the lower specificity due to the sequence length, but utilizes the more exact matching made by a short sequence. The higher AT/GC bias of the short oligonucleotide probes is also avoided. The use of mismatch probes permits the cross-hybridization to be estimated and removed [67], which is not possible with the relative design – at least not when cDNAs are used as probes. Similar designs can obviously be made with ink-jet synthesis and with spotted pre-synthesized oligonucleotides.

Affymetrix' design has more redundancy as well as correction for cross-hybridization. In this way, it should give better results. However, this may not be necessary for relative comparisons. Hughes et al. [74] found that cross-hybridization could be avoided with careful selection of probes, and that one large oligonucleotide was sufficient for relative comparisons. However, they suggest that the use of several probes may be important for determining the absolute concentration of several samples since these measures have a much larger variance.

2.8 Alternative methods

Microarray technologies are not the only methods for performing large genome studies. There are other technologies that can measure the expression of thousands of genes and can be used as substitutes for microarray technologies. The raw data created for these technologies is different and may require different preprocessing than microarray data. However, the methods which are introduced and described in this thesis could equally well be applied to data created with such methods. The main alternatives to microarray will therefore be mentioned in the following sections. These are different from microarrays in that the abundance is determined by sequencing rather than by hybridization.

APEX, which is a method for detecting SNPs with spotted microarrays, is also discussed.

2.8.1 SAGE

Velculescu et al. [183] have developed an alternative approach for gene expression studies, which is called Serial Analysis of Gene Expression (SAGE). The approach is quite different from microarrays since there is no matching of probes and targets. mRNA abundance is rather estimated by gene sequencing and counting. All mRNA strands in a sample are sequenced resulting in a set of sequences stored in a database. The mRNA abundance is then computed by counting the number of times each unique sequence occurs in this set. However, it is not feasible to sequence the full length of the mRNAs in a sample. So only a part of each mRNA strand is used. These parts are later joined into long strands, which are sequenced.

In more detail, cDNAs are first made from the mRNAs in the sample. A slice of 9–14 base pairs, called a sequence *tag*, is cut from each cDNA strand. These tags represent the complete sequences in the further analysis. The tags are split in two pools where a tag in the first pool is concatenated with a tag from the other forming so-called a *ditag*. The ditags are further concatenated with 4 base pair long sequences, which separate the ditags. The final sequence has the following format:

$$\begin{array}{l}
\texttt{CATC UUUUUUUUU VVVVVVVVV CATG UUUUUUUUU VVVVVVVVV CATG} \\
\texttt{GTAC}\ \underbrace{\underbrace{\texttt{YYYYYYYYY}}_{\text{tag 1}}\ \underbrace{\texttt{ZZZZZZZZZ}}_{\text{tag 2}}}_{\text{ditag}}\ \texttt{GTAC}\ \underbrace{\underbrace{\texttt{YYYYYYYYY}}_{\text{tag 3}}\ \underbrace{\texttt{ZZZZZZZZZ}}_{\text{tag 4}}}_{\text{ditag}}\ \texttt{GTAC}
\end{array}$$

The final strands are amplified (i.e., duplicated) by PCR and sequenced. The sequence data is then loaded into a database for further analysis.

Amplification process may change relative concentration of the cDNAs since efficiency of this process depends on the sequence composition. Ditags provide, however, the means for detecting and removing this effect. The probability of creating several occurrences of the same ditag when the tags from the two pools are merged is quite low. A repeated ditag would have to be a result of the amplification. All repeated ditags are therefore removed during sequence analysis.

The details of the sequences analysis is as follows [91]:

1. The ditags are first extracted.

2. All repeated occurrences of the same ditag are removed in order to avoid the bias of the amplification process.

3. The two tags are identified in each ditag as the bases closest to each end.

4. The occurrences of each tag are counted.

5. Each tag is compared to known gene sequences in order to identify the gene, to which the tag corresponds.

One advantage of this approach is that no probes have to be made in advance. Moreover, the gene and its sequence need not to be known prior to the experiment such that the expression of unknown genes can be detected [182]. However, this can also be done with oligonucleotide arrays by synthesizing probes for all possible sequences.

There are several drawbacks with SAGE. Most important is the problem that a tag may not identify a gene uniquely due to the short tag sequence. A tag may therefore map to several genes. This problem is even worsened by the fact that DNA sequences are not random. Some tags are therefore more likely than others. Moreover, the sequencing process may introduce errors such that the incorrect tag sequence may be stored in the database and used in further analysis. Hence, the contribution of an incorrectly sequenced tag will be ascribed to another tag (and gene) than the actual tag in the sample. These and several other issues are discussed in [172, 91]. The sequencing of the concatenated tags still requires a lot of time. So the through-put of this technology is much lower than that of microarrays.

2.8.2 MPSS

A more recent method is Massively Parallel Signature Sequencing (MPSS) [17, 18]. It can be described as a combination of microarray technology and SAGE, but it is closer to SAGE since it relies on sequencing and counting.

The method uses amplification to increase the sensitivity, but it avoids the distortion that may be introduced by this process. This is achieved by attaching a unique tag to each strand in the sample prior to amplification, and loading strands onto microbeads such that strands with identical tags are attached to same microbead. One strand in the original sample is then represented by one microbead.

The strands bound to each microbead are identified by a 16–20 base long signatures (which are similar to tags in SAGE). The sequences of the signatures are found in parallel for all beads using a novel sequencing method. The expression of a gene can then be determined by counting the microbeads with a signature matching a gene.

In more detail, MPSS analysis consists of following steps:

1. A set of tags is initially created. Each tag is 32 bases long and consists of 8 words. A word has 4 bases, and only 8 of 256 ($= 4^4$) possible words are used. These are selected such that they have the same A-T and G-C proportions, and therefore the same ability to hybridize. There are a total of 8^8 different tags.

2. mRNAs in a sample are converted to cDNA, and a tag is attached to each cDNA molecule. The number of different tags is much larger than the number of molecules. So the probability of two cDNA molecules receiving the same tag is very low. The tagged cDNAs are then amplified (i.e., multiplied).

3. Oligonucleotide anti-tags are synthesized onto microbeads such that each microbead has multiple copies of the one anti-tag. The tagged cDNAs and microbeads are allowed to hybridize so that the cDNAs become attached to their matching microbead. A microbead will have approx. $100,000$ copies of the same cDNA strand after the hybridization [18].

4. However, the gene to which the cDNA belongs it is still unknown. The cDNA on the microbeads is therefore partially sequenced by finding a signature. This

is not practical or even feasible with standard sequencing method. A parallel sequencing method is used instead.

A flow cell is used for immobilizing and spreading the microbeads. Signatures of 20 bases are sequenced in 5 iterations. Four bases are read in each iteration. The cDNA strands attached to the microbeads are cleaved with an enzyme resulting in an overhang where one of the double strands is 4 bases longer that the other. Adapters that function as encoders, are applied to the overhangs. An adapter has one end, which matches the overhang and another end, to which a decoding oligonucleotides can bind. Each adaptor tests only one base at a particular position in the overhang. There are 16 different adapter groups. A group consists of 4^3 (= 64) adapters – one for each of the possible sequences when one base is fixed. All of the adaptors in a group encode for the same decoder oligonucleotide. The total set of 1024 (= 16 $\times$ 64) adapters is applied to the microbeads such that a microbead has 4 different adapters – one for each position – attached in equal proportions. Fluorescently labeled decoder oligonucleotides that bind to the adapters are applied iteratively and detected with a laser scanner in the same manner that spots are detected on an microarray, One iteration is made for each of the 16 groups/decoders.

A short overview of the method can be found in [180]. The method is much faster than SAGE and has a more robust system for performing amplification. The signatures are also longer than tags used in SAGE and the method is less plagued with tags that map to multiple genes. However, this problem is not completely avoided as long as only a small part of the gene sequences is used for identifying the gene.

2.8.3 APEX

Arrayed Primer Extension (APEX) [131, 130] is a method for identification of SNPs and is an alternative to Affymetrix' technology for SNP-testing. Pre-synthesized oligonucleotide probes are spotted on microarrays either by using mechanical microspotting or ink-jets. Each probe is designed such that its sequence ends with the base immediately before the position that it should inspect. A sample containing the mutations to be inspected is hybridized on the array, and the DNA binds to their matching probes as with ordinary microarrays.

However, the sample is not labeled. Each probe is instead extended with a single fluorescently labeled nucleotide, which is complementary to the position to be inspected. The nucleotides are labeled with different dyes so that each nucleotide type has one particular dye (A-yellow, C-red, G-green, T-cyan). The base in the sequence can therefore by identified by the color of the probe. The color is determined as with cDNA microarrays by laser scanning the microarray at 4 different wavelengths.

The elongation of the probe is made with DNA polymerases. This enzyme adds nucleotides to the probe strands using the target bound to the probe as a template. The added nucleotide will consequently be complementary to the target strand. However, DNA polymerase will continue to add nucleotides that match the target strand if this processes is not terminated. Nucleotides with different dyes could therefore be added to the probe. This is avoided by using Dideoxynucleotides rather than ordinary nucleotides. These have one end blocked by a hydrogen atom such no new nucleotide can be linked to them. The probe strand is effectively terminated after the addition of such a nucleotide.

Several different mutations can be detected with this technology. The most important of these are base changes, deletions, insertions, and repetitions [174].

Microarray Analysis 3

3.1 Introduction

DNA microarray technology makes large gene expression studies possible, but it generates a vast amount of data about complex biological phenomena. Automated analysis by computers is therefore essential.

Knowledge discovery and data mining have an important role in this respect since they provide computational tools for conducting this analysis. This chapter discusses some of these tools and their application in microarray studies. In particular, clustering and supervised learning of microarrays will be reviewed. Preprocessing is also very important in microarray analysis, and several normalization methods will be presented. Since methods for construction of gene networks are outside the scope of this thesis, they will not be considered here.

The discussion in this chapter will mainly focus on analysis of cDNA microarrays since only this type of data is analyzed in this thesis. However, the analysis of GeneChips is not much different. The difference is mainly in the preprocessing of the array.

3.2 The process

Knowledge discovery is often described as a process with many steps which may be iterated several times. One example is Fayyad et al. [50] who defines a knowledge discovery process with the following steps: Data selection, preprocessing, transformation, data mining, and interpretation. Microarray analysis is no different. However, some steps are particular for this analysis such as normalization, and others are not relevant such as data selection. Microarray analysis passes through the following steps (see Figure 3.1):

1. **Image analysis**: The images from each cDNA microarray are initially converted into ratios, which are log_2-transformed.

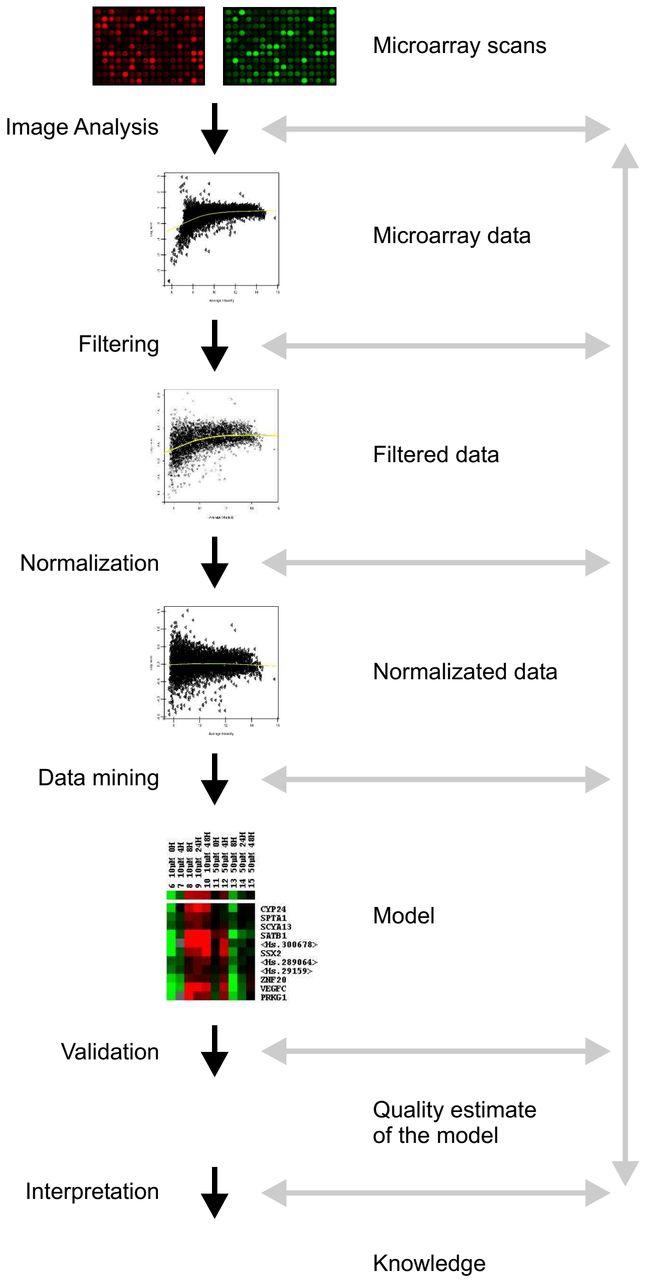


Figure 3.1: *The microarray analysis process.*

2. **Filtering**: The shape of the spots may be inadequate or the expression level may be under the detection limit. Spots with low average intensity often have a larger variance than higher intensity spots. This may be due to insufficient detection for one dye. Such spots must be removed.

3. **Normalization**: Several factors may introduce systematic bias in the result. These biases must be removed.

4. **Data Mining**: A set of arrays is then analyzed with a data mining method. The choice of method obviously depends on the research goal, and this results in a suitable model.

5. **Validation**: The quality of the model is estimated.

6. **Interpretation**: The model should explain the measured data with some certainty and its interpretation should provide new knowledge.

Further preprocessing and transformation may be required besides log_2-transformation, filtering, normalization depending on the data mining method. Missing values (if no ratio can be computed for a spot) may need to be handled. Creation of new features from the ratios with discretization and feature extraction may be necessary.

3.3 Image analysis

A cDNA microarray is read with a laser scanner. This results in two image files corresponding to each of the fluorescent dyes. However, the images need to be quantified into values that can be used for further analysis. This is done by an image analysis program that turns the probe spots on the images into ratios. Such a program detects the region occupied by each spot in the images and estimates the intensity by finding the average or median pixel intensity inside the region. It then computes ratios from the estimated intensities according to Equation 2.1. The ratios are usually log_2-transformed in order to represent induction and repression (i.e., increased or decreased expression level) of a gene on similar scales. This also makes the spot intensities from an image more normally distributed.

Since image analysis is the initial task in any microarray analysis, it has been studied quite extensively. There are many program packages available. Some examples are: GenePix [6], ScanAlyze [47], QuantArray [129], and Scanalytics' IPLAB Microarray Suite [148], which implements the methods of Chen et al. [24]. Yang et al. [195] provides a comparison of several methods.

Most image analysis packages also provide some means for filtering and normalization. However, these solutions are less mature, and these tasks still pose research problems. Separate tools are often used for these problems.

3.4 Normalization

The data extracted from a microarray are very noisy. This is a serious problem in microarray analysis since it may result in false conclusions being drawn from data. The noise should thus be removed as much as possible within affordable limits.

We may distinguish between two kinds of errors in the data:

1. Inherent randomness in samples and measurements, and
2. Systematic bias due to some external factors.

The first category can be reduced with replicates. The mean or median of replicates will be less sensitive to this type of error. The second category can be removed by estimating the influence of the factors and removing their contribution from the measurements. Normalization deals with the last type of error.

3.4.1 Systematic effects

There are many sources in the production of microarrays that may introduce bias. The printing process is by no means perfect, and the quality of the hybridization may affect the result. The following is a list of sources of systematic bias that may be found on a microarray:

- **Labeling effects**: The genes may incorporate dyes Cy3 and Cy5 with different efficiency during the labeling procedure such that the measured intensity signals for one dye becomes generally higher than the intensities for the other [178]. This results in too high or low ratios. Such effects are often visible on microarrays. They may depend on the intensity such that the effects are more apparent for low intensities.

  Moreover, the incorporation efficiency may depend upon the gene sequence such that the bias in the dye intensities may be gene specific. This has been observed by several authors [84, 178].

- **Scanning effects**: The bias in the dye intensities is not only the result of incorporation differences, but also due to the scanning. The ability of the dyes to absorb and emit light may be dissimilar, and detection of two emission frequencies by the scanner may be biased [178].

  Light emitted from fluorescent molecules may be reabsorbed by other fluorescent molecules and result in quenching of the signal. Ramdas et al. [142] found fluorescence quenching on cDNA microarrays. They conducted experiments with diluted oligonucleotides, and found that the range of the actual signals was shorter than the theoretical range. In particular, Cy5-signals were found to be more quenched than Cy3-signals.

  There are also other factors that may introduce bias in the signal range so that the signals of one dye have a shorter range than the signals of the other dye. Finkelstein et al. [52] mention several sources such as photo-bleaching, and improper estimation of background intensity.

- **Printing effects**: The printing process may introduce several biases. There may be variations in the amount of probe cDNA immobilized on the arrays. In particular, the print tips used for deposition cDNA may be different. The amount of cDNA fetched from the print plates and left on the array may be print tip dependent. There may also be variations between each fetch and print iteration. The print tips are cleansed between each iteration, some of the water used in this process may be left in wells on the print plates, and the cDNA in the wells may become diluted. Probe spots with cDNA from the same well may therefore have slightly different concentration depending on when the spot was printed. Print tip effects have been reported by Yang et al. [196] and Finkelstein et al. [54].

  The plates containing the cDNA, which is printed on the glass slides, may introduce variation. For example, the cDNA concentration may be slightly different from plate to plate. This may be due to PCR amplification of cDNA [52]. Plate effects have been detected for 384-well plates using a statistical F-test [53].

- **Hybridization effects**: The target sample may distribute unevenly on the slide. The efficiency of the hybridization reaction can also vary over the slide [153]. This could result in spatial biases. In this case, both global effects over the whole array and local effects just inside a sector/subarray may occur.

  The amount of cDNA applied to the array and environment in the hybridization chamber may affect the result [178].

3.4.2 Methods

Several papers have addressed the issue of normalization. Most of theses center or scale the distribution of the spots on an array. In the first case, the contribution of a source is additive on $\log_2$-scale, and the removal of factor amounts to subtracting the estimated contribution. In the second case, the source is multiplicative on $\log_2$-scale, and the source is removed by dividing on a scaling factor.

Furthermore, we may also consider additive corrections to the dye intensities on the ratio scale (i.e., before $\log_2$-transformation). Such errors are usually assumed to have been removed with background subtraction. However, the background subtraction may not be sufficient, or the procedures for estimating the background may be inadequate.

Centering

There are several different methods for centering. These may be characterized according to which sources of error they take into account. A list of the methods that have been suggested so far is presented in the following.

- **Global normalization**: Plots of the intensity distribution of the dyes reveal whether or not their distributions are shifted. The mean of the intensity signals of one dye is often lower than the mean of the intensity signals of the other. This may yield too low or high $\log_2$-ratios. In this case, we may assume that the means of the dye distributions should be the same such that the $\log_2$-ratio distribution have a mean at 0. This can be achieved by subtraction of the mean $\log_2$-ratio.

  $$\textit{Normalized log}_2\textit{-ratio} = \log_2(Cy5/Cy3) - c \tag{3.1}$$

  Where $c = 1/n \sum \log_2(Cy5/Cy3)$ (assuming that there are n spots on the array. This correction has been widely used (e.g. [6]). Chen et al. [24] devise a more elaborate procedure using an iterative algorithm for estimating the normalization constant C.

- **Intensity dependent normalization**: The shift in the dye signals and the $\log_2$-ratio may depend on the signal intensity. Such effects are often visible in ratio-intensity (RI) plots[1]. An RI-plot is a scatterplot of the $\log_2$-ratio versus the average $\log_2$-intensity of the spots on an array. The average intensity is defined as $I = (\log_2(Cy3) + \log_2(Cy5))/2$. One example is shown in Figure 3.2a where spots with low average intensity are more widely spread, and the shift depends on the intensity.

  This bias can be removed by fitting a normalization function f that depends on the intensity. The normalized ratio of a spot is then computed by subtracting the value of $f(I)$ from the ratio where I is the average intensity of the spot:

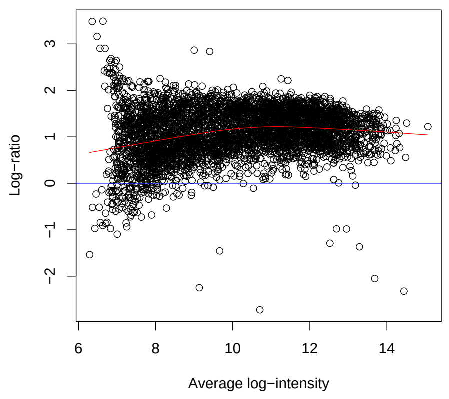
  $$\textit{Normalized log}_2\textit{-ratio} = \log_2(Cy5/Cy3) - f(I) \tag{3.2}$$

  This results in localized shift of the plot. The function $f(I)$ can be estimated with linear regression, but the effects may be nonlinear such that nonlinear regression is more suited. Yang et al. [196] suggest using LOWESS (LOcally Weighted linear regrESSion) [27] for fitting a nonlinear function.

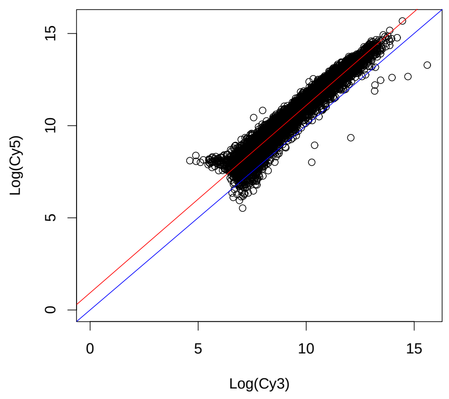
  Some authors perform a similar adjustment fitting regression lines in $\log_2(Cy5)$ vs. $\log_2(Cy3)$ plots (See Figure 3.2b). Finkelstein et al. [54] use an iterative linear regression for removing outliers in such plots. The use of $\log_2(Cy5)$ vs. $\log_2(Cy3)$ plots predates the RI-plots in normalization of microarrays, but the RI-plot is preferable. The fact that normalization is dependent on the average intensity is also made more evident. Moreover, an RI-plot is just a 45° rotation and scaling of the $\log_2(Cy5)$ vs. $\log_2(Cy3)$ plot, and spot artifacts more easily seen in this plot.

- **Logshift normalization**: Kerr et al. [84] introduce an alternative to intensity normalization. RI-plots of some arrays may have a slight curvature where the low intensity spots have a lower $\log_2$-ratio than the rest. Kerr et al. suggest that

[1]Ratio-intensity plots are also known as MA-plots.



(a) RI-plot



(b) log(Cy5) vs. log(Cy3)-plot

Figure 3.2: *Two different scatterplots of the same microarray data.* Both plots display intensity-dependent normalization functions. The red curve in the RI-plot is the LOWESS-fit. The red line in log(Cy5) vs. log(Cy3)-plot is found with linear regression.

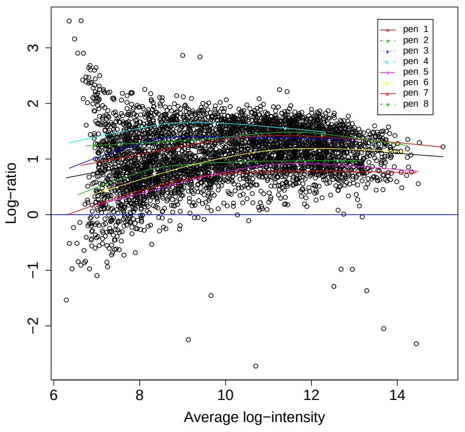


Figure 3.3: *A scatterplot with intensity dependent normalization curves for each print tip.*

the effect could arise as a result of an additive difference between the two dyes on the ratio-scale (e.g., the estimate of background signal, which is subtracted from the dye signals, may be improper). They adjust the signal with term s and a log-ratio is then computed as:

$$NLR_g(s) = \log(Cy5_g - s) - \log(Cy3_g + s) \tag{3.3}$$

for gene g, and s is estimated as:

$$s = \underset{t}{\operatorname{argmin}} \sum_g \left| NLR_g(t) - \underset{g}{\operatorname{median}}\, NLR_g(t) \right| \tag{3.4}$$

- **Dye-swapping**: This is an alternative for correcting the shift between the two dyes. The two samples are hybridized together on two microarrays. However, labeling is reversed for one of the slides. For example, sample a is labeled with $Cy3$ and sample b is labeled with $Cy5$ on one array. On the other array, sample a is labeled with $Cy5$ and sample b is labeled with $Cy3$. If the errors are of similar magnitude on the arrays (i.e., $c_1 = c_2$), they will be removed by subtracting one of the $\log_2$-ratios for a spot from the other. The average ratio can be found as

$$\begin{aligned} & \frac{1}{2}\left(\left(\log_2 \frac{Cy3_1}{Cy5_1} + c_1\right) - \left(\log_2 \frac{Cy3_2}{Cy5_2} + c_2\right)\right) \\ = & \frac{1}{2}\left(\log_2 \frac{Cy3_1}{Cy5_1} + \log_2 \frac{Cy5_2}{Cy3_2}\right) = \frac{1}{2}\left(\log_2 \frac{a_1}{b_1} + \log_2 \frac{a_2}{b_2}\right) \end{aligned} \tag{3.5}$$

  Dye swapping has been introduced by several authors. Kerr et al. [85] show how microarrays can be analyzed with ANOVA[2], and demonstrate in particular how dye-swapping can be modeled with a latin square design (e.g., [188]). Yang et al. [196] examine the usefulness of dye-swapping, and found that this method was appropriate when the arrays had similar normalization function.

  The bias, which is removed with dye-swapping, may be specific for each spot on the array. An important feature of this method is that gene-dye interactions can be corrected. However, the method assumes that $\log_2$-ratios on arrays have the same additive bias and scale.

- **Print tip and plate normalization**: As mentioned, the print tips used to deposit the cDNA during printing may be a source of bias. If such bias occurs, normalization can be done individually for each print tip. The spots deposited by the same print tip are collected and a normalization constant or function dependent on the average intensity is estimated. The normalized $\log_2$-ratios are computed as:

$$\textit{Normalized } \log_2\textit{-ratio} = \log_2(Cy5/Cy3) - f_p(I) \tag{3.6}$$

[2]ANOVA means "ANalysis Of VAriance". See e.g., [188] for more information on this topic.

where p identifies the print tip. This type of normalization was introduced by Yang et al. [196]. They use LOWESS to fit each f_p(I). An example is given Figure 3.3 where the normalization function for each print tip is shown on same microarray data as in Figure 3.2. Finkelstein et al. [53] give a similar approach where plate effects (as described on page 29) are also considered. They suggest a method for detecting this type of bias. An ANOVA test (e.g., [188]) is used where the plates and the print tips are considered as factors. Corrections to the dye intensity signals are computed separately for each print tip and each plate. The shift due to the two dyes is corrected with linear regression after these other corrections have been applied.

Their method assumes that effects are linearly related such that a normalization function can be fitted independently for each effect. Any interactions between print tips, plates, and intensity will not be taken into account. The method used by Yang et al. does not make such assumptions. However, the normalization function, which is fitted to the data, depends on more variables since all effects are considered at once. This normalization function must output a value for each input combination of these variables, and this output value must be estimated from the data with the same (or similar) input combination. Unfortunately, less data are available for each combination since there are more combinations, but the number of spots is still the same. The data are consequently relatively more sparse for this function than for a function with only one variable. The risk of overfitting the function is higher, and the assumption that most genes are not regulated, may be less likely or not valid at all for some combinations.

Notice that it is important that the gene probes are placed at random on the array. Otherwise, the observed print tip or plate effects may be caused by the localization of the spots and not by a source of error.

- **Spatial normalization**: The print tips create sectors on the array. All spots in a sector are deposited by the same pin, and sectors are spatially separated on the glass surface. The print tip bias may therefore be considered a spatial effect. However, other spatial effects may also be observed on the glass surface. Patterns may be observed inside sectors, across sectors, and over whole arrays. Such biases can be corrected by fitting a LOESS function (which is a successor to LOWESS and allows the function to depend on several variables):

$$\text{Normalized } \log_2\text{-Ratio} = \log_2(Cy5/Cy3) - f(I, p, R, C) \tag{3.7}$$

where R is the row and C is the column of the spot in the sector p. Alternatively one may fit a function $f_p(I, R, C)$ for each sector. As mentioned there is a problem that the normalization function is based on less spots as more factors are considered. One may, however, assume that some factors do not interact and decompose the model, e.g. $f(I, p, R, C) = f(I) + f_p(R, C)$.

It may also be difficult to discern spatial and print tip effects. The spatial effect is to some degree confounded with the print tip effects due to the spatial extension of the print tip sectors.

While there is some awareness of such spatial effects [52], no work addressing spatial normalization has been published at the time of writing.

Scaling

The variance of the log-ratios may be different from array to array or inside arrays. Yang et al. [196] distinguish between two different kinds of scaling. Scaling may be performed within an array or between arrays. In the first case, they adjust the variance of each print tip sector with a maximum likelihood estimate. In the second case, the variance of each array is adjusted such that it is the same over all arrays. In both cases, the scaling factors are found with a maximum likelihood estimate. It is also possible to devise scaling strategies between other groups, e.g., the plates. However, less work has been done on scaling than on centering.

Other methods

The method described above applies to cDNA microarrays where two samples are hybridized on the same array. When only one sample is used per slide such as Affymetrix' GeneChips, there is no dye-shift to correct. However, the mean log-intensity of the spots may still vary between arrays, and the arrays may need to be centered. The correction must be computed over arrays, in this case, such that every array has the same center. Some of the methods devised for cDNA microarrays may be used. For example, Beissbarth et al. [9] use global normalization to center two arrays. Hartemink et al. [68] use arrays with so-called spiking controls which are spots with predetermined abundance. The arrays are centered according to the spikes using a maximum a posteriori approach. They assume that the log-intensity of each spot is normally distributed with a different variance of each spot. Zien et al. [198] take a similar approach, but apply a maximum likelihood approach instead. Moreover, they do not use spikes and assume that most genes are not regulated or there is similar number of up and down regulated genes on each array. Kepler et al. [83] take a more advanced strategy. They propose a model where both the mean and the variance of the measurements depend on the average intensity. These factors are estimated with LOWESS.

3.4.3 Assumptions

In order to normalize two or more samples there must be some features that should be similar in all samples. Observed differences in these features can then be used for adjusting the samples. Most of the normalization methods discussed above adjust the intensity distributions such that the means become similar. This results in a centering of the $\log_2$-ratio distribution at 0. Global normalization performs this correction. The intensity, print tip, and plate normalization essentially make the same correction. However, in the latter methods, the corrections are made on a local rather than global basis. For example, the mean intensities of the two dyes are made similar for each print tip or plate.

The biological assumption behind the normalization methods is that most genes are not regulated, and should thus have identical intensity in all samples. A similar assumption is to expect the number and the magnitude of up and down regulated genes to be approximately the same in all samples. However, the normalization will not be better than the assumption, and the assumption may be flawed. Especially, on a small array where only a part of the genome is printed.

One alternative is to use so-called housekeeping genes. These are genes responsible for the maintenance of the cell and are supposed to be expressed at constant level. If these are not known in advance they can be identified.

Tseng et al. [178] present one method where housekeeping genes are selected according to the rank difference of the dye intensities. The genes are ranked separately for each dye. A gene is selected if its ranks for the two dyes differ less than a specified threshold. However, such selection will depend upon the unnormalized data. A gene, which is actually differentially expressed may be selected and used in the normalization, if the difference between the unnormalized dye intensities is small.

There are no universally stable genes [52]. The constant expression of the housekeeping genes may actually depend on the cell type from which the sample is taken and stress conditions applied to the sample. Housekeeping genes that are constantly expressed in one cell type do not need to be constantly expressed in another. Hence, it may be difficult to use such an approach if many different cell types and stress conditions are used in the same experiment.

Spiking controls are a better alternative. These are spots with a probe sequence, which is dissimilar to any naturally occurring cDNA sequence in the organism. This can be achieved by selecting cDNA from another organism (or by using specifically designed oligonucleotides). The complementary strands of these probes are mixed with the target samples such that the abundance is fixed. The intensity and $\log_2$-ratio of the spiking controls is predetermined in other words. Spiking controls have been used by several authors [46, 153].

Notice that the normalization techniques described above can also be used with both housekeeping genes and spiking controls. In this case, spots corresponding to the housekeeping gene or the controls are used in the normalization. Furthermore, the spiking control can also be used for assessing the quality of the normalization techniques when other assumptions are used.

3.5 Cluster Analysis

After preprocessing, a set of microarrays can be analyzed with a data mining tool. Cluster analysis is the most popular method in microarray analysis and has been used in many studies. This is an exploratory technique that discovers groups of objects in a data set. The groups are formed on the basis of similarities in the objects and are often called clusters (cf. the name of the method). A clustering algorithm will typically start with a certain number of groups and try to place the objects into these groups such that some criterion is minimized. Alternatively, it may discover a hierarchy of groups (i.e., a taxonomy). In this case, each single object forms a group at the bottom of the hierarchy, and the groups higher up in the hierarchy are formed by merging two or more groups from a lower level.

There are many different clustering methods and several of these have been applied to microarrays. Eisen et al. [48] used Hierarchical Clustering (e.g. [164, 165]). A particular feature of their approach is the way the clustering is visualized. They accompany the dendrogram made by a hierarchical cluster with a so-called heat map with the color intensity scaled according to the ratios. Negative ratios are displayed as green and positive ratios as red. Tamayo et al. [175] applied Self Organizing Maps [86] to microarray data. Tibshirani et al. [177] compared several different clustering methods including K-means [97, 105]. However, hierarchical clustering is by far the most widely used technique in microarray studies.

Two different problems are considered with cluster analysis on microarrays; either genes or samples are clustered. In the first case, groups of similarly expressed genes are found. In the second case, groups of similar samples (or test conditions) are discovered. In both situations, one cDNA microarray is prepared for each test condition, and a test sample is hybridized together with a reference sample. The reference sample must be biologically equivalent for all microarrays. This can be achieved by using the same sample on all microarrays or by using biologically similar material taken from different individuals. One example of the last case are cancer studies where a tumor sample and a non-cancerous sample are taken from each patient and the pair of samples is hybridized together (see e.g.,[92, 103]) on an array. The non-cancerous samples are supposed to be equivalent. Any differences in these samples are patient specific and can be ignored.

The data given to a clustering algorithm are often represented in a table or a matrix where the objects to be clustered correspond to the rows and the attributes of these objects correspond to the columns. We may represent the $\log_2$-ratio in this manner. When genes are clustered, the test conditions are considered as attributes of the genes, and each row in the table describes a gene, and each column corresponds to a test condition. When the test samples are clustered, the matrix is transposed, and the genes are used as attributes (see Figure 3.4).

3.5.1 Clustering of genes

As mentioned in Section 2.3, transcription of genes is regulated by transcription factors that bind to the binding sites in the promoter region of the genes. Genes with the same binding sites will be controlled by the same transcription factor, and the expression of them will often be similar. However, there may be several binding sites in the promoter region, and genes may share some, but not all binding sites. The expression may therefore be different. One often distinguishes between *co-expressed* genes, which have similar expression profiles and *co-regulated* genes, which are regulated by the same transcription factors or by the same biological stimuli.

Clustering has been used for discovering co-expressed genes. The response in cells to some external stimulus is considered in this type of study. It is usually measured as time sequence where each test condition corresponds to a time point. One example is

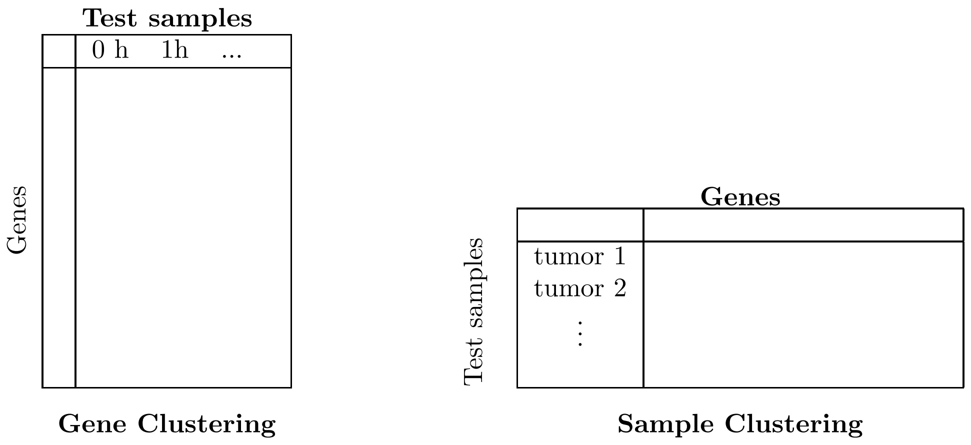


Figure 3.4: Two ways to cluster microarray data

Iyer et al. [76]. They examined the response in human fibroblast cells to fetal bovine serum when this serum was reapplied after it had been removed for a period of 48 hours. They identified 10 major clusters with co-expressed genes. However, the test conditions do not need to be time points. They can also be the responses to several experimental conditions. Several different time sequences can be also considered in the same clustering. Spellman et al. [169], for example, studied several time responses in yeast which were triggered by different conditions.

The co-expressed gene clusters can be further analyzed in two different ways:

- **Identification of regulatory binding sites**: Co-expressed genes may be co-regulated and may have a common transcription factor binding site. However, these binding sites are often unknown. One may try to locate a site by searching through the promoter regions of these genes and identify a sequence that is similar in all promoters.

  One of the first studies of this kind was made by Tavazoie et al. [176]. They used K-means to cluster the data. They then searched with a sequence analysis program for common sequence motifs in the promoter regions of the genes in each cluster. A similar study was made by Vilo et al. [184]. They used another sequence algorithm and focused more on the validation of the discovered motifs. Two scores were applied to select the motifs: A silhouette value and a pattern score. The silhouette value quantified the quality of an expression cluster. The pattern score was defined for a pattern π and an expression cluster C as the inverse of the probability that π would occur in the promoter regions of the genes within C. A simulation with random data was used for setting the thresholds for the selection of patterns. Similar patterns were also post-processed with clustering.

- **Assignment of functional annotations**: The genes have been identified in many organisms, but they have not been characterized. For example, Gene Ontology database [29] contained (as of 07 Dec 2001) a process annotation, i.e., a description of biological objective, for only 2639 genes in *Saccharomyces cerevisiae* (yeast) while there are approximately 6000 genes in this genome [110].

  A very plausible reason for the similar behavior of co-expressed genes is that they take part in the same cellular process. The simultaneous behavior occurs as a result of the regulation of the process. The need for the gene products is increased or decreased as the activity of the process changes. Co-expressed genes identified by hierarchical clustering have also been shown to participate in the same process. (e.g., [48]). Consequently, if we assume that this hypothesis holds, we may predict the function of an uncharacterized gene by looking at the characterized genes occurring in same cluster.

  This assumption has been used in many studies for suggesting the function of uncharacterized genes — or at least for proposing hypotheses about the function (see e.g., [26, 76]). In Iyer et al. [76], data were hierarchically clustered, and gene clusters were selected visually by inspecting the dendrogram and the heat map. They hypothesized that the uncharacterized genes in several of the clusters had the same function as the characterized genes in these clusters. Cho et al. [25] also used hierarchical clustering, but applied a binomial test to select clusters with a high proportion of one particular function.

  Walker et al. [187] made a similar study, but did not use clustering. Furthermore, they did not assign a functional description to the genes, but they looked instead for genes that were involved in prostate cancer. Some of the genes were already known to be involved in this disease, and if one of these known genes and another gene had similar expression profiles, this other gene was also assumed to be associated with the disease. The similarity of the expression profile was determined with a statistical test that tested for independence between pairs of genes. A gene was assumed to be involved if null hypothesis of this test was rejected (i.e., the genes were not independent).

  However, there are some difficulties with this approach to predict gene function. These have been recognized by Shatkay et al. [156] who point out that:

  1. Genes may have more than one function such that a strict assignment to one cluster may be inappropriate,
  2. Functionally related genes may be anti-co-regulated in their expression profiles such they that end up in different clusters, and
  3. Co-expressed genes may not share the same function.

  Hence, the assumption that co-expressed genes may share function may not hold, and even if it holds, clustering may not be the best method to predict the function.

3.5.2 Clustering of samples

Tumors are not identical – even when they occur in the same organ, and patients may need different treatments depending on their particular subtype of cancer. Identification of tumor subgroups is therefore important for diagnosis and design of medical treatments.

Most medical classification systems for tumors are currently based on clinical observations and the microscopical appearance of the tumors. However, these observations are not informative with regard to the molecular characteristics of the cancer. This is unfortunate as the mechanism underlying the tumorgenic process is found on the molecular level. Mutations in the DNA sequence may result in changes in gene expression [3, 139], and these changes may disrupt the normal cell processes such that the cell becomes cancerous. Since microarrays can measure such changes, they may reveal, which kinds of defects there may be and provide insight into what goes wrong in a cancerous cell. In particular, they may reveal the mutagenetic variations that constitute the basis for different tumor subtypes.

Several studies have clustered tumor samples derived from various patients and measured with microarrays. This includes, amongst others, studies of B-cell lymphoma [2], breast cancer [166, 137], and cutaneous malignant melanoma [12].

However, while clustering of genes is a more or less "solved" problem, clustering of samples is more difficult. This is due to *the curse of dimensionality* – the data become dramatically sparser in multidimensional space as the number of dimensions increases. The production of microarrays is still quite expensive, and the number of samples that are affordable to analyze in a study is limited. A study with 50-100 samples is considered very large[3]. The number of genes, which are the attributes (or dimensions) in this kind of study, will be much larger than the number of samples. The problem is therefore under-determined. Numerous clusterings may be discovered in the data. In particular, different subsets of the genes may yield quite different groups. The relevant genes must therefore be selected before the cluster analysis such that the "right" clustering is obtained.

Unfortunately, we do not know the "right" clustering (this is what we seek in the first place), and the discovered clustering depends on the selected genes. So there is no external criterion for selecting the relevant genes. Still, one may assume that the major variation in the data yields the right clustering and find the genes that contribute the most to this variation. Such genes can be found with principle component analysis (PCA) (also known as singular value decomposition). This technique has been applied to microarray data by [177, 73]. Alternatively, one may try to fit data into a low-dimensional space with either PCA or multidimensional scaling such as Bittner et al. [12].

However, these may not yield the best results and some new methods have been introduced. Gene Shaving [71, 70] is a novel technique, which selects subsets of genes. It sorts the genes according to the first principle component of the samples (the genes

[3]Number of microarrays that were used in the mentioned studies are: 128 microarrays for B-cell lymphoma study [2], 78 for breast cancer [166, 137], and 31 for cutaneous malignant melanoma [12].

are considered to be the dimensions) and removes (shaves off) the genes with the least contribution until a subset with an optimal size is found. This size is determined with a GAP-statistic. The data is orthogonalized and the procedure is repeated. Several orthogonal subsets are found where each subset yields a different clustering of the samples. Another approach has been introduced by Xing and Karp [194]. They use a mixture model and make several assumptions about the gene behavior to select the genes.

3.6 Supervised learning

The objective of supervised learning is to find a function that maps an object to one of several classes based on some observed attributes of the object. A supervised learning algorithm takes a set of objects with class labels as training data and finds a function that maps each of the objects to the correct class. The discovered function can later be used to classify new objects without any class label. Supervised learning is also known as classification since objects are mapped to discrete classes. Sometimes the class labels are real numbers. In this case, the problem of finding a function is called regression.

There are many different supervised methods that have been applied to microarrays. Support vector machines [181] have been applied by Furey et al. [58] and Brown et al. [20, 19]. This method finds a plane that separates maximally between two classes. A class prediction for an object is then made according to which side of the plane the object is located. Dudiot et al. [39] use discriminant analysis (e.g. [78]), which also finds a separating plane. They also used K-nearest neighbor (e.g., [118]), which makes a prediction based on the classes of the K most similar objects and decision trees [141, 16], which builds a tree that is used for prediction. In the last case, a class is assigned to an object by traversing the tree from the root down to a leaf class where a prediction is made. At each internal node an attribute is examined and one of the child nodes is visited according the value, which the object has for this attribute. Rule learning methods such as rough sets [134] have been used by Hvidsten et al. [75]. Such methods find a set of rules that are used for predicting the class of an object.

Classification and clustering are closely related. Clustering may be used for classification. In this case, the class of an object is predicted on basis of the cluster, in which the object occurs. For example, we may assume that an object belongs to the most frequently observed class in its cluster.

However, there are important differences. Clustering attempts to find groups of similar objects, but the groups may not be suitable for predicting the class. Supervised learning on the other hand tries to discriminate between the instances from different classes in order to predict the class of an object. It makes a model such as a discriminating plane or a set of rules, which is optimized for separating between the instances of dissimilar classes. Thus a supervised method is more appropriate for this type of problem, and will usually have better prediction performance than a clustering

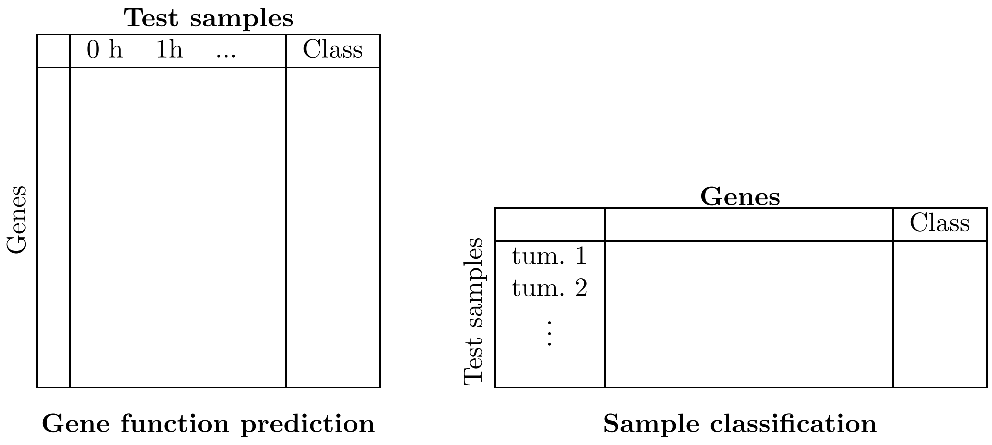


Figure 3.5: Two ways to classify microarray data

method.

Moreover, the criticism of Shatkay et al. (which is mentioned on page 38) is less relevant to supervised methods. A supervised method can handle anti-co-regulated behavior and predict several classes at once such that all functional categories of a gene are predicted (see e.g., [75]).

There are two types of problems that can be addressed by supervised learning. Either genes or samples can be classified just as in clustering. The difference is mainly that we now have an extra class attribute as shown in Figure 3.5.

3.6.1 Learning to classify genes

There are several studies on predicting gene function where supervised methods have been used. Brown et al. [20, 19] applied support vector machines (SVM) to the data from Eisen et al. [48], and predicted 5 functional categories from the MIPS[4] ontology [57]. They also compared the performance of SVMs to several other supervised methods and showed that SVMs outperformed the other methods on this data set.

More recently, Hvidsten et al. [75] used Rough Sets [134] to develop classifiers for 16 functional categories from the Gene Ontology (GO) consortium's biological process ontology [29]. Uncharacterized genes were then classified according to these categories. The processes were selected such that a classifier was built for a process if at least 10 genes were annotated with this process or one of the subprocesses.

Pavlidis et al. [132] continued the work of Brown et al. [20, 19]. They used both gene expression data and phylogenetic profiles, and studied three different methods for combining these different types of data in an SVM:

[4]MIPS is an abbreviation for "Munich Information center for Protein Sequences".

1. The data set is either initially merged, and an SVM is trained on the merged data set.

2. Separate kernel[5] functions are used for each data set. The sum of these kernels is then used in a single SVM.

3. An SVM is trained on each data set separately and the discriminating values produced by the SVMs are added together.

The second approach yielded the best performance in their experiments.

Still, none of these studies attempted to induce a complete classifier. GO or MIPS ontologies contain hundreds of classes, but only subsets of the classes were selected. The ontologies also define relationships between the classes such that they have a structure[6]. This structure was largely ignored. The classes were simply chosen so that they were unrelated. However, no considerations were made to the structure in the learning and the prediction phase, since the learning algorithms could not take the structure into account.

Unfortunately, valuable information is lost when the ontological structure is not considered, and this lost information may result in suboptimal predictions. New learning algorithms that learn in the ontology are consequently needed. This problem will be considered in Chapters 6-10.

3.6.2 Learning to classify samples

Discovery of tumor subtypes is important in cancer studies. However, it is equally important to predict such subtypes from microarray samples so that this information can be applied in the diagnosis of cancer patients.

In particular, the disease may manifest itself earlier at the molecular level than at the clinical level. Prediction of tumor subtype and other clinical parameters from gene expression data may therefore enable earlier detection and treatment of the disease. This may again increase the survival rate. Moreover, the genes, whose expression levels are associated with the tumor subtypes, are largely unknown. A better understanding of the cancer could be achieved if these genes are identified.

These problems can be addressed by supervised learning by training a classifier on the microarray samples. The class (or decision) attribute is in this case tumor subtypes or other clinical parameters such as the survival time. By inspecting the

[5]A support vector machine finds a hyperplane that creates the largest margin between the objects of two different classes. This means an SVM in its simplest form is limited to finding linear boundaries. A kernel function is a similarity measure between two (information) vectors and provides a means to escape this limitation. Such an function defines an implicit mapping from one feature space to another feature space where the features in the new space may be non-linear functions of the features in the original space. So, when a separate kernel function is used for each data set, a individual mapping is created to for each set. However, only one hyperplane is found. For more information on SVMs, see e.g., [33].

[6]The GO ontology forms a directed acyclic graph, and the MIPS ontology is a tree (See Chapter 6 for more details).

classifier one may also determine the genes that have the most impact on a clinical parameter.

Just as in cluster analysis of samples, the main problem in this type of study is the curse of dimensionality. The number of samples is much smaller than the number of genes. Hence, a feature selection method is required. The selection problem is nevertheless easier than the corresponding problem in clustering since a reference variable, the class attribute, is available. Feature selection for supervised learning has also received a lot of attention in machine learning and pattern recognition. Some surveys are [15, 34].

Golub et al. [60, 162] made the first classification study of tumors for microarrays. They successfully discerned between two different types of leukemia. Initially a set of significant genes was selected using a permutation test. The selected genes were then used in a "weighted voting" algorithm introduced by Golub et al. Briefly, each gene is used individually as a classifier. Given a new sample, each gene produces a vote in favor of one class, and the weighted sum of votes is used for classifying the sample. Note, however that the algorithm is not really novel, as pointed out by Dodoit et al. [39]. It is merely an instance of the diagonal linear discriminant algorithm.

Several studies have compared the performance of various classification methods in this problem domain. Ben-Dor et al. [10] tested K nearest neighbors (K-NN), SVMs, boosting of decision stumps, and a clustering-based approach. They selected genes according to a TNoM (Threshold Number Of Misclassification) score. They found that the feature selection method improved the prediction accuracy. None of the methods out-performed the rest on both data sets used in their study. The best method depended on the data set.

Dudiot et al. [39] compared several linear discriminant methods, K-NN, decision trees (CART), and bagging/boosting of decision trees. Gene selection was done according to the ratio of between-class and within-class sums of squares for each gene. They obtained good results for most of the methods with diagonal linear regression and K-NN yielding the best results.

The Naive Bayes algorithm is applied by Keller et al. [82]. They proposed a maximum likelihood score for selecting genes. The approach was compared to the method of Golub et al. [60, 162] on several data sets.

Furey et al. [58] used an SVM and claimed that SVMs are less affected by the curse of dimensionality. They demonstrated that feature selection does not yield better results for SVMs on some data sets. However, this seems at least inconsistent with the results of Ben-Dor et al., which used some of the same data sets and found that the feature selection gave better results for SVMs.

Tree harvesting is a feature extraction procedure, which has been introduced by Hastie et al. [69]. The genes are first clustered with hierarchical clustering, and the mean log ratio within each cluster is found for each sample. The clusters are then used as features in an iterative regression approach where one feature is added to the model in each iteration. This approach can obviously be used with other learning methods as well.

More information on supervised learning of microarray samples can also be found in the book collection from CAMDA-2000 conference [93]. This collection contains several papers on applying supervised learning methods to the data of Golub et al. [60, 162].

Rough Set Theory

4

4.1 Introduction

This chapter gives an introduction to rough set theory (RST) [133, 134, 135] which constitutes a foundation for the work that will be presented in the following chapters. This introduction only describes the fundamental parts of RST. More specialized methods that are often used in combination with RST (e.g., discretization) will be introduced later when they are needed.

RST is a methodology for dealing with uncertainty in data, and it is most often used for developing supervised learning approaches. As a methodology for developing classifiers, RST has several appealing qualities:

- The classifier produced with RST consists of a set decision rules. These are quite close to natural language and are easily interpreted by a user.
- Many supervised methods can only produce binary classifiers. Rough set theory, on the other hand, can deal with classification problems with more than two classes.
- RST handles inconsistencies in the training data. In particular, it can deal with objects (e.g., genes) that are labeled with conflicting classes. Such inconsistencies may occur as a result of noise in the data or by lack of information.
- In some classification problems, it may be necessary to predict more than one class for each object. However, most supervised learning methods predict only one class per object. RST methods, on the other hand, are well-suited for dealing with such problems. In particular, its facilities for handling inconsistencies may be used for multiple decision classes as well.
- Rough set based learning finds a minimal set of attributes for which a classifier can be built. This allows the classifier to focus on attributes that are relevant for

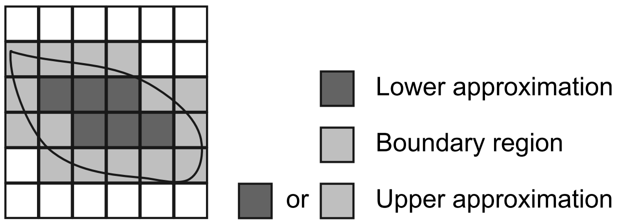


Figure 4.1: *A rough set.* The figure shows the partition of elementary sets and the contour of a set X. The lower approximation of X is display with dark gray, and the boundary region with light gray. The upper approximation consists of lower approximation and the boundary region.

the classifications problem and discard noisy, redundant and otherwise irrelevant attributes. An RST classifier will therefore be less sensitive to such problems.

4.2 Rough sets: The idea

The main idea behind rough set theory is that the membership of some elements with regard to a set may be uncertain such that the set can only be described by an approximation. More precisely, RST assumes that there is a *universe* of *objects* and some information associated with each object. The objects that cannot be told apart given only the available information are called *indiscernible.* A set of indiscernible objects is called an *elementary* set, and the elementary sets constitute the least granules that can be distinguished with the available information. Using the elementary sets as building blocks, one may estimate how a set of objects can be characterized in terms of the available information. Since an elementary set may be both partially inside and outside of this set, such a characterization may not be exact. Hence, a rough set is given in terms of a lower approximation and an upper approximation (see Figure 4.1) where the lower contains all elementary sets completely inside and the upper contains all elementary sets partially or completely inside.

4.3 Information systems

Rough set theory is usually developed from an *information system.*

Definition 4.1 (Information system). The tuple $\mathcal{A} = \langle U, A \rangle$ is an information system if

1. U is a non-empty finite set of objects, called the *universe*, and

Object	***0H-15m***	***15m-30m***	***30m-1H***	***1H-2H***	***2H-4H***
o_1	up	up	up	const	const
o_2	up	up	up	down	down
o_3	down	down	down	up	up
o_4	up	up	up	down	down
o_5	down	up	up	down	up
o_6	const	const	const	down	down
o_7	down	down	down	up	up
o_8	up	down	down	down	down
o_9	const	down	down	down	down
o_{10}	down	up	up	up	up
o_{11}	down	down	up	up	up

Table 4.1: *An information system* $\mathcal{A} = \langle U, A \rangle$. In this particular case, the objects in U are genes, and the attributes in A describe the change in expression level between two adjacent time points.

2. A is a set of *attributes* which describe the objects, and each attribute a in A is a function $a : U \to V_a$ where V_a is the set of values that an object may take for attribute a. ■

An information system is basically a table where each column corresponds to an attribute and each row to an object. One example is shown in Table 4.1. The set of attribute-value pairs that are associated with an object is called an information vector.

Definition 4.2 (Information vector). Given an information system $\mathcal{A} = \langle U, A \rangle$ and an object $x \in U$, the information vector is

$$Inf_A(x) = \{\langle a, a(x) \rangle \mid a \in A\}$$ ■

The objects that have the same information vector are indiscernible. Such objects are identified by an *indiscernibility relation.*

Definition 4.3 (Indiscernibility relation). Let $\mathcal{A} = \langle U, A \rangle$ be an information system. The indiscernibility relation is defined as

$$\begin{aligned} IND(A) &= \{\langle x, y \rangle \in U \times U \mid Inf_A(x) = Inf_A(y)\} \\ &= \{\langle x, y \rangle \in U \times U \mid a(x) = a(y), \text{for all } a \in A\} \end{aligned}$$ ■

The indiscernibility relation $IND(A)$ is an equivalence relation and partitions the universe into equivalence classes, which are called elementary sets in RST terminology.

Definition 4.4 (Elementary set). Let $\mathcal{A} = \langle U, A \rangle$ be an information system. An elementary set of $x \in U$ is

$$[x]_A = \{y \in U \mid \langle x, y \rangle \in IND(A)\}$$ ■

The set of all elementary set ($\{[x]_A \mid x \in U\}$), is called the quotient set and is denoted as $U/IND(A)$.

The definitions of the lower and the upper approximation of a set X are based on the elementary sets. The lower approximation contains an object x if its elementary set $[x]_A$ is a subset of X. The upper approximation contains an object x if the intersection of its elementary set $[x]_A$ and X is non-empty (i.e., $[x]_A$ and X overlap partially). It is therefore certain that the objects in the lower approximation are in X. The objects in the upper approximation, on the other hand, may belong to X, but may also belong to the complement of $U - X$.

Definition 4.5 (Approximations). Given $B \subseteq A$ where $\mathcal{A} = \langle U, A \rangle$, the upper and the lower approximations of a set $X \subseteq U$ is given as follows:

- **B-lower approximation**: $\underline{B}X = \{x \mid [x]_B \subseteq X\}$
- **B-upper approximation**: $\overline{B}X = \{x \mid [x]_B \cap X \neq \emptyset\}$ ■

Note $\underline{B}X$ is always a subset of $\overline{B}X$. The set of objects that are in $\overline{B}X$, but not in $\underline{B}X$, constitutes the *boundary region*

Definition 4.6 (Boundary region). Given $B \subseteq A$, the boundary region of X is $BND_B(X) = \overline{B}X - \underline{B}X$. ■

If $\underline{B}X = \overline{B}X$ ($BND_B(X) = \emptyset$), then X is defined exactly and is called *crisp*. Otherwise, the set is called *rough*.

Example 4.1. Table 4.1 constitutes an information system $\mathcal{A} = \langle U, A \rangle$ where $U = \{o_1, \ldots, o_{11}\}$ and $A = \{$`0H-15m, 15m-30m, 30m-1H, 1H-2H, 2H-4H`$\}$. The indiscernibility relation, formed by A, has the following elementary sets:

$$U/IND(A) = \{\{o_1\}, \{o_2, o_4\}, \{o_3, o_7\}, \{o_5\}, \{o_6\}, \{o_8\}, \{o_9\}, \{o_{10}\}, \{o_{11}\}\}$$

This means for instance that objects o_2 and o_4 are indiscernible, while object o_5 can be discerned from the rest.

If $X = \{o_1, o_2, o_3\}$, then the lower and the upper approximations are $\underline{A}X = \{o_1\}$ and $\overline{A}X = \{o_1, o_2, o_3, o_4, o_7\}$, respectively. The boundary region $BND_A(X)$ consists consequently of $\{o_2, o_3, o_4, o_7\}$. ■

The upper and the lower approximations have several properties:

1. $\underline{B}X \subseteq X \subseteq \overline{B}X$
2. $\underline{B}\emptyset = \overline{B}\emptyset = \emptyset$, $\underline{B}U = \overline{B}U = U$
3. $\overline{B}(X \cup Y) = \overline{B}X \cup \overline{B}Y$
4. $\underline{B}(X \cup Y) \supseteq \underline{B}X \cup \underline{B}Y$
5. $\underline{B}(X \cap Y) = \underline{B}X \cap \underline{B}Y$
6. $\overline{B}(X \cap Y) \subseteq \overline{B}X \cap \overline{B}Y$

7. $X \subseteq Y$ implies $\underline{B}X \subseteq \underline{B}Y$ and $\overline{B}X \subseteq \overline{B}Y$
8. $\underline{B}(-X) = -\overline{B}(X)$, $-X = U - X$
9. $\overline{B}(-X) = -\underline{B}(X)$
10. $\underline{B}(\underline{B}X) = \overline{B}(\underline{B}X) = \underline{B}X$
11. $\overline{B}(\overline{B}X) = \underline{B}(\overline{B}X) = \overline{B}X$

An important issue in rough set theory is whether some of the attributes in an information system are redundant and may be removed without reducing the discernibility of the objects. A minimal subset of attributes that maintains the original indiscernibility relation is called a *reduct*.

Definition 4.7 (Reduct). A reduct is a minimal subset $B \subseteq A$ such that $IND(B) = IND(A)$. ■

Example 4.2. The information system in Table 4.2 has one reduct $B = \{$`0H-15m, 15m-30m, 30m-1H, 1H-2H`$\}$. The attribute is `2H-4H` thus redundant and may be removed. ■

4.4 Decision systems

An information system treats all attributes similarly. In a classification problem, however, the objects are labeled with decision classes which are to be predicted from the other attributes. An information system is thus not directly applicable to such problems, but may be extended to a *decision system*.

Definition 4.8 (Decision system). A decision system $\mathcal{A} = \langle U, A, d\rangle$ is an information system where

- U is the universe of objects,
- A is a set of conditional attributes, and
- d is a decision attribute, which does not occur in A. It is a function $d : U \to V_d$ where $V_d = \{d_1, \ldots, d_n\}$ is the set of decision classes that may be assigned to the objects. ■

For each decision class $c \in V_d$, we may create a set $X_c = \{x \in U \mid d(x) = c\}$ with all objects that belong to the class. The upper and the lower approximations may be applied to these sets in order to determine whether the class of an object can be predicted uniquely from the condition attributes. For example, the objects in the lower approximation $\underline{A}X_c$ belong all to class c and may be discerned from the objects in $U - X_c$. The class of these objects can therefore be predicted uniquely from the conditional attributes. The objects in the boundary region $BND_A(X_c)$, on the other hand, belong to several classes, and it is not possible to assign a unique decision class to these objects.

Object	*0H-15m*	*15m-30m*	*30m-1H*	*1H-2H*	*2H-4H*	*Process Name*
o_1	up	up	up	const	const	cell proliferation
o_2	up	up	up	down	down	cell proliferation
o_3	down	down	down	up	up	cell proliferation
o_4	up	up	up	down	down	cell adhesion
o_5	down	up	up	down	up	cell adhesion
o_6	const	const	const	down	down	cell adhesion
o_7	down	down	down	up	up	cell growth & maintenance
o_8	up	down	down	down	down	cell growth & maintenance
o_9	const	down	down	down	down	intracellular protein traffic
o_{10}	down	up	up	up	up	transport
o_{11}	down	down	up	up	up	cytoplasmic transport

$$\underbrace{\hspace{8cm}}_{A}\quad\underbrace{\hspace{4cm}}_{d}$$

Table 4.2: *A decision system* $\mathcal{A} = \langle U, A, d\rangle$. This table contains the same objects/-genes and (conditional) attributes as Table 4.1. The only difference is the decision attribute **Process** that describes the function of the genes.

The union of the lower approximations over all decision classes is called the *positive region.*

Definition 4.9 (Positive region). Let $\mathcal{A} = \langle U, A, d\rangle$ be a decision system, then the positive region is $POS_A(d) = \bigcup_{c\in V_d} \underline{A}X_c$ where $X_c = \{x \in U \mid d(x) = c\}$. ∎

A decision system is called *consistent* if $POS_A(d) = U$, and in this case a unique decision class may be predicted for every object. If $POS_A(d) \neq U$, the decision system is *inconsistent.*

Example 4.3. Table 4.2 is identical to the information system in Table 4.1 except for the decision attribute **Process**. $X_{\texttt{cell adhesion}} = \{o_4, o_5, o_6\}$ is the set of objects labeled with `cell adhesion`. The lower and the upper approximations of this set are $\underline{A}X_{\texttt{cell adhesion}} = \{o_5, o_6\}$ and $\overline{A}X_{\texttt{cell adhesion}} = \{o_2, o_4, o_5, o_6\}$. Hence, the class of o_4 cannot be determined uniquely from the conditional attributes. The positive region is $POS_A(\textbf{Process}) = \{o_1, o_5, o_6, o_8, o_9, o_{10}, o_{11}\}$, which obviously is only a subset of U. Hence, the system is inconsistent. ∎

One may also find reducts for decision systems. However, it is not necessary to discern between objects that are labeled with the same classes in this case. So the condition in Definition 4.7 may be relaxed such that reducts are found relative to the decision attribute. A *decision-relative reduct* is defined as follows.

Definition 4.10 (Decision-relative reduct). Let $\mathcal{A} = \langle U, A, d\rangle$ be a decision system. A decision-relative reduct is a minimal subset $B \subseteq A$ such that $POS_B(d) = POS_A(d)$. ∎

However, this definition is quite weak. It requires that the decision classes of the objects in $POS_A(d)$ must be determined uniquely from the attributes in B. So, t

Object	***a***	***b***	***d***
x_1	1	1	d_1
x_2	2	1	d_2
x_3	3	2	d_1
x_4	3	2	d_3
x_5	3	3	d_2
x_6	3	3	d_3

Elementary set	***Classes of*** $[\mathbf{x}]_{\mathbf{A}}$	***Classes of*** $[\mathbf{x}]_{\mathbf{B}}$
x_1	d_1	d_1
x_2	d_2	d_2
x_3	d_1, d_3	d_1, d_2, d_3
x_4	d_1, d_3	d_1, d_2, d_3
x_5	d_2, d_3	d_1, d_2, d_3
x_6	d_2, d_3	d_1, d_2, d_3

Table 4.3: *The decision system in Example 4.4.* The first subtable presents the decision system $\mathcal{A} = \langle U, A, d\rangle$. The second subtable provides the classes that will be predicted for an object when attributes in $A = \{a, b\}$ or attributes in $B = \{a\}$ are used. For each object x, the table lists the classes of the objects in $[x]_C$ (where $C = A$ or $C = B$). Note that this set of classes is same as the generalized decision $\partial_C(x)$.

is sufficient if the decision system is consistent. However, it does not guarantee that objects in the boundary region are not assigned more classes when the decision system is inconsistent.

Example 4.4. Consider the decision system in Table 4.3. The indiscernibility relation on $A = \{a, b\}$ partitions the universe into the following elementary sets:

$$U/IND(A) = \{\{x_1\}, \{x_2\}, \{x_3, x_4\}, \{x_5, x_6\}\}$$

The objects in the last two elementary sets have different classes, so that they are not in the positive region. Hence, the positive region $POS_A(d)$ is equal to $\{x_1, x_2\}$. $B = \{a\}$ is subset of A, and the indiscernibility relation $IND(B)$ defines the following partition:

$$U/IND(B) = \{\{x_1\}, \{x_2\}, \{x_3, x_4, x_5, x_6\}\}$$

This means that that the positive region $POS_B(d)$ is also $\{x_1, x_2\}$. So, B is a decision-relative reduct of A (since $POS_B(d) = POS_A(d)$). However, objects x_3 and x_4 would only be assigned to d_1 and d_3 with A (See the second subtable of Table 4.3). Objects x_5 and x_6 would similarly be assigned just to d_2 and d_3. With B, on the other hand, these objects are assigned to all of the classes (d_1, d_2, d_3). Hence, the reduct B will introduce more classes for the objects in the boundary region. ■

This problem may be corrected by creating a new decision system $\mathcal{A}^* = \langle U, A, \partial_A\rangle$ with a *generalized decision* attribute and then find a decision-relative reduct on this system.

Definition 4.11 (Generalized decision). Let $\mathcal{A} = \langle U, A, d\rangle$ be a decision system. A generalized decision for object $x \in U$ is $\partial_A(x) = \{d(y) \mid y \in [x]_A\}$. ■

The generalized decision attribute makes a new compound class for each object x in the boundary region. This class consists of the original classes of all of objects in elementary set $[x]_A$. Since the indiscernibility relation is an equivalence relation, we have that $[x]_A = [y]_A$ for all $y \in [x]_A$. All objects in $[x]_A$ therefore have the same generalized decision class. Hence, the new decision system $\mathcal{A}^* = \langle U, A, \partial_A \rangle$ created from the generalized decision attribute is consistent such that every object in U belongs to the positive region $\mathcal{A}^*$.

Example 4.5. Objects o_2, o_3, o_4, and o_7 do not belong to the positive region in Table 4.2. The generalized decisions for these objects are:

$$\partial_A(o_2) = \partial_A(o_4) = \{\texttt{cell proliferation, cell adhesion}\}$$
$$\partial_A(o_3) = \partial_A(o_7) = \{\texttt{cell proliferation, cell growth \& maintenance}\}$$

The decision system $\langle U, A, \partial_A \rangle$ created by this generalized decision has only one decision-relative reduct:

$$B = \{\texttt{0H-15m, 15m-30m, 30m-1H, 1H-2H}\}$$

Note that this is the same reduct as we found in Example 4.2. So, in this particular case, $IND(B) = IND(A)$ (However, this is purely coincidental). ■

4.5 Decision rules

From a decision system, we can find decision rules and produce a classifier that can classify new objects. Decision rules describe the concepts defined by the decision classes using the conditional attributes and have the form

$$\alpha \rightarrow \beta$$

where the antecedent α consists of descriptors such as $\langle a, v \rangle$ made from the attributes in A, and the conclusion β consists of descriptors made from the decision attribute d.

Definition 4.12 (Rule syntax). Let $\mathcal{A} = \langle U, A, d \rangle$ be a decision system. The language of decision rules $\mathcal{L}(A, \{d\})$ consists of all formulae $(\alpha \rightarrow \beta)$ where $\alpha \in \mathcal{L}(A)$ and $\beta \in \mathcal{L}(\{d\})$. A formula γ is in $\mathcal{L}(B)$ if:

1. $\gamma = \langle a, v \rangle$, $a \in B$, and $v \in V_a$
2. $\gamma = \alpha \wedge \beta$ and $\alpha, \beta \in \mathcal{L}(B)$
3. $\gamma = \alpha \vee \beta$ and $\alpha, \beta \in \mathcal{L}(B)$
4. $\gamma = \neg\alpha$ and $\alpha \in \mathcal{L}(B)$ ■

Example 4.6. Consider the decision system in Table 4.3. Some of formulae that may be constructed from this decision system are:

$$\begin{array}{ll} \langle a,1\rangle \wedge \langle b,1\rangle & \langle a,1\rangle \wedge \langle b,2\rangle \\ \langle a,1\rangle \vee \langle b,1\rangle & \neg\langle a,1\rangle \\ \neg\langle a,1\rangle \wedge \langle b,3\rangle & \langle a,1\rangle \wedge \langle b,1\rangle \rightarrow \langle d,d_1\rangle \end{array}$$ ■

A rule in $\mathcal{L}(A, d)$ has the following semantics with regard to the decision system.

Definition 4.13 (Rule semantics). Let $\mathcal{A} = \langle U, A, d\rangle$ be a decision system. A formula α in $\mathcal{L}(A, d)$ (or $\mathcal{L}(A)$ or $\mathcal{L}(\{d\})$) has the semantics $[\![\alpha]\!]_{\mathcal{A}}$ which is defined inductively as:

1. $[\![\langle a, v\rangle]\!]_{\mathcal{A}} = \{x \in U \mid a(x) = v\}$
2. $[\![\alpha \wedge \beta]\!]_{\mathcal{A}} = [\![\alpha]\!]_{\mathcal{A}} \cap [\![\beta]\!]_{\mathcal{A}}$
3. $[\![\alpha \vee \beta]\!]_{\mathcal{A}} = [\![\alpha]\!]_{\mathcal{A}} \cup [\![\beta]\!]_{\mathcal{A}}$
4. $[\![\neg\alpha]\!]_{\mathcal{A}} = (U - [\![\alpha]\!]_{\mathcal{A}})$
5. $[\![\alpha \rightarrow \beta]\!]_{\mathcal{A}} = (U - [\![\alpha]\!]_{\mathcal{A}}) \cup [\![\beta]\!]_{\mathcal{A}}$ ■

Example 4.7. Consider the decision system in Table 4.3 again. The semantics of the formulae in Example 4.6 is:

$$\begin{array}{rr}
[\![\langle a,1\rangle \wedge \langle b,1\rangle]\!]_{\mathcal{A}} = \{x_1\} & [\![\langle a,1\rangle \wedge \langle b,2\rangle]\!]_{\mathcal{A}} = \emptyset \\
[\![\langle a,1\rangle \vee \langle b,1\rangle]\!]_{\mathcal{A}} = \{x_1, x_2\} & [\![\neg \langle a,1\rangle]\!]_{\mathcal{A}} = \{x_2, x_3, x_4, x_5, x_6\} \\
[\![\neg \langle a,1\rangle \wedge \langle b,3\rangle]\!]_{\mathcal{A}} = \{x_5, x_6\} & [\![\langle a,1\rangle \wedge \langle b,1\rangle \rightarrow \langle d,d_1\rangle]\!]_{\mathcal{A}} = U
\end{array}$$ ■

An object $x \in U$ *satisfies* a rule $\alpha \rightarrow \beta$ if $x \in [\![\alpha \rightarrow \beta]\!]_{\mathcal{A}}$ and it *satisfies the antecedent* of the rule if $y \in [\![\alpha]\!]_{\mathcal{A}}$. A rule *covers* all of the objects that satisfy its antecedent, i.e., the objects in $[\![\alpha]\!]_{\mathcal{A}}$. We denote the objects covered by a set of rules $RS \subseteq \mathcal{L}(A, d)$ by

$$Cov_{\mathcal{A}}(RS) = \{x \in U \mid x \in [\![\alpha]\!]_{\mathcal{A}} \text{ and } (\alpha \rightarrow \beta) \in RS\}$$

The rules in RS that have a single descriptor with class c in their conclusions are denoted as

$$RS_c = \{(\alpha \rightarrow \beta) \in RS \mid \beta = \langle d, c\rangle\}$$

The support of a rule is the number of objects that match the both the antecedent and the conclusion. It is an estimate of the number of objects that is predicted correctly by the rule.

$$Support(\alpha \rightarrow \beta) = |[\![\alpha]\!]_{\mathcal{A}} \cap [\![\beta]\!]_{\mathcal{A}}|$$

If a rule has more than one descriptor in the conclusion, we may also consider the support of the rule with respect to each class. The support is in this case defined as

$$Support_c(\alpha \rightarrow \beta) = |[\![\alpha]\!]_{\mathcal{A}} \cap [\![\langle d, c\rangle]\!]_{\mathcal{A}}|$$

where $\langle d, c\rangle$ is in the conclusion β.

Example 4.8. Assume the decision system in Table 4.2 and consider the following rules:

$$\begin{aligned} r_1 &= \langle \texttt{0H-15m, up}\rangle \wedge \langle \texttt{15m-30m, up}\rangle \wedge \langle \texttt{30m-1H, up}\rangle \wedge \langle \texttt{1H-2H, down}\rangle \rightarrow \langle \texttt{Process, cell prolif.}\rangle \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vee \langle \texttt{Process, cell adhesion}\rangle \\ r_2 &= \langle \texttt{0H-15m, up}\rangle \wedge \langle \texttt{15m-30m, up}\rangle \rightarrow \langle \texttt{Process, cell prolif.}\rangle \end{aligned}$$

The semantics of the first rule with regard to the decision system in Table 4.2 is $[\![r_1]\!]_{\mathcal{A}} = U$. Hence, all objects satisfy this rule. Rule r_2, on the other hand, has the semantics $[\![r_2]\!]_{\mathcal{A}} = U - \{o_4\}$ since object 4 satisfies the antecedent, but not the conclusion of this rule. So the rule is not consistent with the decision system.

The coverage of the rules are $Cov_{\mathcal{A}}(\{r_1\}) = \{o_2, o_4\}$ and $Cov_{\mathcal{A}}(\{r_2\}) = \{o_1, o_2, o_4\}$. Both rules have a support of two since

$$Support(\{r_1\}) = |\{o_2, o_4\}| = 2 \text{ and } Support(\{r_2\}) = |\{o_1, o_2\}| = 2.$$

Since r_1 has several descriptors in conclusion, we may also consider the support of each class. In this case both have support of 1:

$$Support_{\texttt{cell prolif.}}(\{r_1\}) = |\{o_2\}| = 1 \text{ and } Support_{\texttt{cell adhesion}}(\{r_1\}) = |\{o_4\}| = 1$$

■

4.6 Learning decision rules

Rules are found in two different ways in RST; either by finding reducts first and then creating rules (with possibly several descriptors in the conclusion) from the reducts or by finding two rule sets that cover the lower and the upper approximations of each class.

4.6.1 The reduct approach

In the first approach pioneered by Skowron and his co-workers [158, 160], a new decision system $\mathcal{A}^* = \langle U, A, \partial_A \rangle$ is created with a generalized decision attribute as explained in Section 4.4. Decision-relative reducts are then found from the new system and rules are created from the reducts.

Finding a minimal reduct is unfortunately NP-hard [161]. So reducts are usually found with by using some heuristic. Typically, a genetic algorithm (see e.g., [116]) is used. Such algorithms are based on an analogy with biological evolution. The algorithm maintains a population (a set) of individuals where each individual contains a chromosome that determines its properties. The population has a fixed size and passes through generations such that the individuals in the population become gradually more fit. In each generation some of the individuals are selected and allowed to reproduce by combining their chromosomes. Some individuals are also removed in order to make room for the new individuals. The probability that an individual will reproduce or be removed from the population is related to its fitness. When a genetic algorithm is used for finding reducts, a chromosome represents a subset of attributes in A. The fitness is based on the number of attributes in the chromosome and the ability of these attributes to discern between the objects with different generalized decisions. For more details, see Wroblewski [192] and Vinterbo and Øhrn [185].

An alternative to a genetic algorithm is Johnson's greedy set cover algorithm [77]. This selects a set of attributes iteratively. In each iteration the attribute that discerns between the largest number of objects with different generalized decision is selected and added to the attribute set. The algorithm terminates when no more objects can be discerned by the remaining attributes. For more details see [127]

When a reduct B has been found, a set of rules RS_B can be created as follows:

$$\begin{aligned} Ant_B(x) &= \bigwedge_{a \in B} \langle a, a(x) \rangle \\ Con_B(x) &= \bigvee_{c \in \partial_B(x)} \langle d, c \rangle \\ Rul_B &= \{Ant_B(x) \to Con_B(x) \mid x \in U\} \end{aligned}$$

Example 4.9. In Example 4.5, we found the reduct $B = \{\texttt{0H-15m}, \texttt{15m-30m}, \texttt{30m-1H}, \texttt{1H-2H}\}$ for the decision system in Table 4.2. The following rules can be created from this reduct:

1. $\langle$0H-15m, u$\rangle \wedge \langle$15m-30m, u$\rangle \wedge \langle$30m-1H, u$\rangle \wedge \langle$1H-2H, c$\rangle \to \langle$Process, cell prolif.$\rangle$ (o_1)
2. $\langle$0H-15m, u$\rangle \wedge \langle$15m-30m, u$\rangle \wedge \langle$30m-1H, u$\rangle \wedge \langle$1H-2H, d$\rangle \to \langle$Process, cell prolif.$\rangle$ $\vee \langle$Process, cell adhesion$\rangle$ (o_2,o_4)
3. $\langle$0H-15m, d$\rangle \wedge \langle$15m-30m, d$\rangle \wedge \langle$30m-1H, d$\rangle \wedge \langle$1H-2H, u$\rangle \to \langle$Process, cell prolif.$\rangle$ $\vee \langle$Process, cell growth$\rangle$ (o_3,o_7)
4. $\langle$0H-15m, d$\rangle \wedge \langle$15m-30m, u$\rangle \wedge \langle$30m-1H, u$\rangle \wedge \langle$1H-2H, d$\rangle \to \langle$Process, cell adhesion$\rangle$ (o_5)
5. $\langle$0H-15m, c$\rangle \wedge \langle$15m-30m, c$\rangle \wedge \langle$30m-1H, c$\rangle \wedge \langle$1H-2H, d$\rangle \to \langle$Process, cell adhesion$\rangle$ (o_6)
6. $\langle$0H-15m, u$\rangle \wedge \langle$15m-30m, d$\rangle \wedge \langle$30m-1H, d$\rangle \wedge \langle$1H-2H, d$\rangle \to \langle$Process, cell growth$\rangle$ (o_8)
7. $\langle$0H-15m, c$\rangle \wedge \langle$15m-30m, d$\rangle \wedge \langle$30m-1H, d$\rangle \wedge \langle$1H-2H, d$\rangle \to \langle$Process, intra. p. traf.$\rangle$ (o_9)
8. $\langle$0H-15m, d$\rangle \wedge \langle$15m-30m, u$\rangle \wedge \langle$30m-1H, u$\rangle \wedge \langle$1H-2H, u$\rangle \to \langle$Process, transport$\rangle$ (o_{10})
9. $\langle$0H-15m, d$\rangle \wedge \langle$15m-30m, d$\rangle \wedge \langle$30m-1H, u$\rangle \wedge \langle$1H-2H, u$\rangle \to \langle$Process, cyto. transp.$\rangle$ (o_{11})

The values *up*, *down*, and *const* are represented here as *u*, *d*, and *c*, respectively. The objects that are covered by each rule are shown in parentheses. ■

If more than one reduct is found, we may create a rule set for each reduct. Each rule set may be used as a classifier on its own. However, a more accurate classifier can usually be obtained by combining the rules obtained from each reduct.

4.6.2 The covering approach

In the second approach, which is used in the LERS system [64, 65], a set of *certain* rules CRS and a set of *possible* rules PRS are found. These rules have only one descriptor in the conclusion and are created independently for each decision class c. For each class c the certain rules CRS_c, which predict c, cover only objects in lower approximation such that $Cov_{\mathcal{A}}(CRS_c) = \underline{A}X_c$. Hence, the rules only predict the decision class when an object certainly belongs to the class. The possible rules cover all objects in the upper approximation such that $Cov_{\mathcal{A}}(PRS_c) = \overline{A}X_c$ for each class c where PRS_c denotes the possible rules predicting c. Thus an object may satisfy antecedent of several rules at once leading to several class predictions.

The rules are found with a *covering* algorithm (also known as a separate-and-conquer algorithm [59]) in this approach. Such an algorithm creates one rule at the time by searching through a hypothesis space consisting of conjunctions of descriptors. When a rule is found, the objects covered by it are removed and if not all objects have been covered yet, the algorithm continues searching for another rule. One example of this kind of algorithm is given in Section 8.5.1. Certain rules are created with it by setting the positive set $\mathcal{P}$ to $\underline{A}X_c$ and the negative set $\mathcal{N}$ to $U - \underline{A}X_c$ ($\gamma = 1$). Possible rules are similarly found with $\mathcal{P} = \overline{A}X_c$ and $\mathcal{N} = U - \overline{A}X_c$. Note that the algorithm used in the LERS system differs slightly from the one in Section 8.5.1. However, the difference is not important here. For the full details see [65].

Example 4.10. We may create the following certain rules from the decision system in Table 4.2:

1.	⟨1H-2H, const⟩	→ ⟨Process, cell adhesion⟩	(o_1)
2.	⟨30m-1H, up⟩ ∧ ⟨1H-2H, down⟩ ∧ ⟨2H-4H, up⟩	→ ⟨Process, cell adhesion⟩	(o_5)
3.	⟨0H-15m, const⟩ ∧ ⟨15m-30m, const⟩	→ ⟨Process, cell adhesion⟩	(o_6)
4.	⟨0H-15m, up⟩ ∧ ⟨15m-30m, down⟩	→ ⟨Process, cell growth⟩	(o_8)
5.	⟨0H-15m, const⟩ ∧ ⟨15m-30m, down)⟩	→ ⟨Process, intra. p. traffic⟩	(o_9)
6.	⟨15m-30m, up⟩ ∧ ⟨30m-1H, up⟩ ∧ ⟨1H-2H, up⟩	→ ⟨Process, transport⟩	(o_{10})
7.	⟨15m-30m, down⟩ ∧ ⟨30m-1H, up⟩ ∧ ⟨1H-2H, up⟩	→ ⟨Process, cytop. transport⟩	(o_{11})

■

Example 4.11. The following possible rules can be found from the decision system in Table 4.2:

1.	⟨0H-15m, up⟩ ∧ ⟨15m-30m, up⟩	→ ⟨Process, cell proliferation⟩	(o_1,o_2,o_4)
2.	⟨0H-15m, down⟩ ∧ ⟨15m-30m, down⟩	→ ⟨Process, cell proliferation⟩	(o_3,o_7)
3.	⟨30m-1H, up⟩ ∧ ⟨1H-2H, down⟩	→ ⟨Process, cell adhesion⟩	(o_4,o_5)
4.	⟨0H-15m, const⟩ ∧ ⟨15m-30m, const⟩	→ ⟨Process, cell adhesion⟩	(o_6)
5.	⟨0H-15m, down⟩ ∧ ⟨15m-30m, down⟩	→ ⟨Process, cell growth⟩	(o_3,o_7)
6.	⟨0H-15m, up⟩ ∧ ⟨15m-30m, down⟩	→ ⟨Process, cell growth⟩	(o_8)
7.	⟨0H-15m, const⟩ ∧ ⟨15m-30m, down)⟩	→ ⟨Process, intra. p. traffic⟩	(o_9)
8.	⟨15m-30m, up⟩ ∧ ⟨30m-1H, up⟩ ∧ ⟨1H-2H, up⟩	→ ⟨Process, transport⟩	(o_{10})
9.	⟨15m-30m, down⟩ ∧ ⟨30m-1H, up⟩ ∧ ⟨1H-2H, up⟩	→ ⟨Process, cytop. transport⟩	(o_{11})

■

Note that finding certain rules is not strictly necessary. It is sufficient to find only possible rules since the possible rules PRS cover all objects that that are covered by the certain rules CRS. Moreover, the objects that are covered by CRS will only be covered by a single rule in PRS such that only one class is predicted for these objects. To see this, consider an arbitrary class c in V_d. The difference between the two is that the possible rules cover the objects in the boundary region of X_c while the certain rules do not. However, neither the certain nor the possible rules cover the negative region $U - \overline{A}X_c$. This region $(U - \overline{A}X_c)$ is a superset of the union of the lower approximations of the other decision classes $(\bigcup_{d \in V_d - \{c\}} \underline{A}X_d)$. The possible rules PRS_c will therefore not cover objects that certainly belong to the other classes, and as this holds for any c, the objects in the positive region will only be assigned a unique class. This is demonstrated in Example 4.11 where, for instance, o_1 satisfies the antecedent of rule 1, o_5 satisfies the antecedent of rule 3, o_6 satisfies the antecedent of rule 4, etc.

4.7 Prediction

When a set of rules RS has be learned, it may be used as a classifier for predicting the class of previously unseen objects. However, the rules may have a conflicting conclusion. This happens with possible rules and with the rules that are created from a generalized decision and have a disjunctive conclusion. This occurs also when an ensemble of rule sets is used together as single classifier where each rule set is created from a different reduct. In this case, the rules originating from the different reducts may have different conclusions.

Voting is a procedure for resolving such conflicts. For an object x, the prediction of the class can be made in the following manner:

1. The set of rules $RS(x)$ that covers x is determined.

$$RS(x) = \{(\alpha \to \beta) \in RS \mid x \in [\![\alpha]\!]_{\mathcal{A}}\}$$

2. If $RS(x) = \emptyset$, no classification can be made. In this case, different actions may be taken. One possibility is to predict the most frequent class in the training data. Another is to predict a predetermined *fallback* class set by the user. Such actions are typically specified through parameters set by the user.

3. The certainty of each class in V_d is computed and the class with the largest certainty is chosen. The certainty is computed as follows:

   (a) Each rule is allowed to cast a number of votes in favor of the class in its conclusion. This number is typically equal to the support of the rule defined as:

   $$votes(\alpha \to \beta) = support_c(\alpha \to \beta)$$

   If a rule has more than one descriptor in the conclusion, we consider the support of each class separately.

   (b) The total number of votes $votes(c)$ for each class c is determined:

   $$votes(c) = \sum_{r \in RS_c(x)} votes_c(r)$$

   where $RS_c(x) = \{(\alpha \to \beta) \in RS(x) \mid \beta = \langle d, c\rangle\}$

(c) Certainty of class c is then computed as:

$$certainty(c, x) = \frac{votes(c)}{\sum_{e \in V_d} votes(e)}$$

Note that the $certainty(c, x)$ is an approximation of the probability $P(c|x)$.

Example 4.12. Assume that the rules in Example 4.11 are applied for predicting the class of object x, which has the following information vector.

$$\begin{aligned} Inf_A(x) = \{ &\langle \texttt{0H-15m, up} \rangle, \langle \texttt{15m-30m, up} \rangle, \langle \texttt{30m-1H, up} \rangle, \\ &\langle \texttt{1H-2H, up} \rangle, \langle \texttt{2H-4H, down} \rangle \} \end{aligned}$$

Only two rules cover this object. These are rules 1 and 8. Unfortunately, the conclusions of these rules are different so that two different classes are predicted. There is consequently a conflict that needs to be resolved by voting procedure described above.

The voting procedure will proceed as follows: Rule 1 has a support of 2 since the objects o_1 and o_2 in Table 4.2 match both the antecedent and the conclusion. The support of rule 8 is 1 since only object o_{10} satisfies both the antecedent and the conclusion. This means that

$$\begin{aligned} votes(\texttt{cell proliferation}) &= 2 \\ votes(\texttt{transport}) &= 1 \end{aligned}$$

since there are no other rules that cover x. The certainty of these two classes is:

$$\begin{aligned} certainty(\texttt{cell prolif.}, x) &= \frac{votes(\texttt{cell prolif.})}{votes(\texttt{transport}) + votes(\texttt{cell prolif.})} = \frac{2}{3} \\ certainty(\texttt{transport}, x) &= \frac{votes(\texttt{transport})}{votes(\texttt{transport}) + votes(\texttt{cell prolif.})} = \frac{1}{3} \end{aligned}$$

Hence, `cell proliferation` has the largest certainty, and it is therefore predicted. ■

4.8 The ROSETTA system

The reduct approach, which was described in Section 4.6.1, has been implemented in a system called ROSETTA [126, 88]. ROSETTA is a rough set toolbox for analysis of data and provides many tools for creating rough set classifiers. It has been applied on a wide range of domains and obtained a large body of users. By April, 2003, it had been downloaded approximately 4700 times since its first release in summer 1997.

Rosetta runs under MS Windows and provides both a user-friendly graphical user interface (GUI) and a command-line interface. The system can also be compiled and run under UNIX. However, only the command-line interface is available in this case.

The system supports most parts of the KDD-process:

- It provides several facilities for importing and selecting data. In particular, it has an ODBC-interface (Open Database Connectivity) which allows the system to connect to and extract data from various database systems.

- It offers some tools for preprocessing. Missing values in data set can be "repaired" with a completion algorithm (also known as an imputation algorithm). Such an algorithm examines a decision system, and if an object is missing a value for an attribute, it attempts to predict the missing value and assigns the predicted value to the object. Rosetta has several different completion algorithms

  Rough set methods can only handle discrete values. Rosetta has therefore several discretization algorithms that can convert real values into discrete values.

- Several algorithms for finding reducts have been implemented in Rosetta. These include an exhaustive search algorithm, a genetic search algorithm, and a greedy search algorithm (based Johnson's greedy set cover). It has also an algorithm for finding dynamic reducts [8, 7], and an algorithm for creating 1R classifiers [72] (which consist of rules with only one descriptor in the antecedent). An algorithm for generating rules from reducts is also provided.

- Prediction of classes is supported through the voting procedure in Section 4.7. The algorithm implemented in Rosetta provides many options that may be adjusted by the user.

- Rosetta supports some evaluation strategies. A classifier can be evaluated by splitting the data set into two sets so that training is done on one set and evaluation is performed on the other set. Evaluation can also be done with leave-one-out or n-fold cross-validation. The system applies performance measures such as accuracy and AUC. (More details about these measures and the evaluation strategies are given in Section 5.2.4).

In the next chapter, Rosetta is used to build classifiers for gastric tumors.

Learning to Classify Microarray Samples

5

Most of this chapter has been published in Midelfart et al. [114]. Some parts are also presented in Nørsett et al. [124].

5.1 Introduction

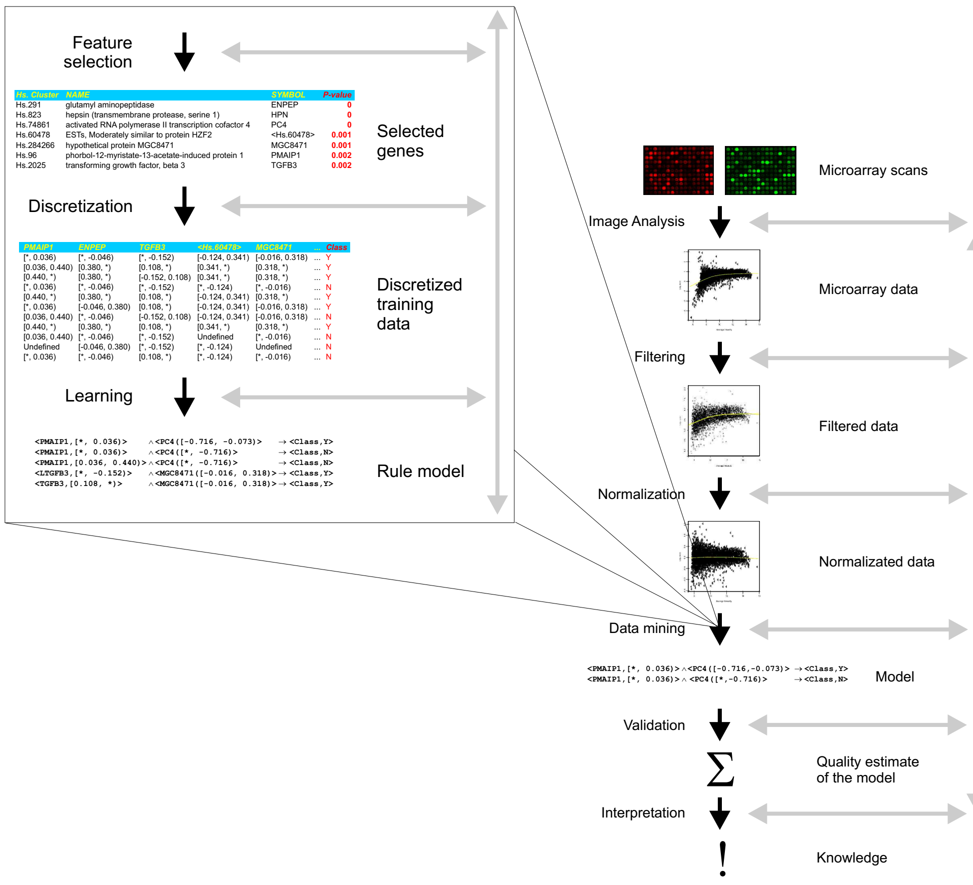
As we discussed in Sections 3.5.2 and 3.6.2, classification of microarray samples (or experiments) is important in diagnosis and treatment of cancer. Predictions based on gene expression data may, in particular, determine tumor subtype and other clinical parameters at an earlier stage than clinical examination. This may allow earlier detection and treatment of the disease, which again may increase the survival rate.

It appears that although there is a general consensus that supervised learning is an appropriate approach to diagnostic analysis of gene expression, there is no comprehensive methodology developed yet. We propose formulating this problem in the framework of Rough Set Theory (RST) and introduce a general methodology based on the rough set system Rosetta [126].

The methodology is shown in Figure 5.1. It contains several steps besides the ones that usually occur in microarray analysis. After image analysis, filtering, and normalization, a feature selection method is applied to the normalized data. This identifies genes that discriminate significantly between the decision classes, and only these genes are used in the further analysis.

Note that this step is required since the size of the universe and the number of conditional attributes have wrong proportions. The cost of producing an array is substantial even with microspotting. An array may cost 100–1,000 US dollars. At the same time, a microarray may hold 2,000–40,000 genes. Hence, the number of microarrays (i.e., objects) will be much smaller than the number of genes (i.e., attributes) in this kind of study.

This means that one may find thousands of decision-relative reducts in the data. However, many of these reducts will only be artifacts of the data and may not have



Hs. Cluster	NAME	SYMBOL	P-value
Hs.291	glutamyl aminopeptidase	ENPEP	0
Hs.823	hepsin (transmembrane protease, serine 1)	HPN	0
Hs.74861	activated RNA polymerase II transcription cofactor 4	PC4	0
Hs.60478	ESTs, Moderately similar to protein HZF2	<Hs.60478>	0.001
Hs.284266	hypothetical protein MGC8471	MGC8471	0.001
Hs.96	phorbol-12-myristate-13-acetate-induced protein 1	PMAIP1	0.002
Hs.2025	transforming growth factor, beta 3	TGFB3	0.002

PMAIP1	ENPEP	TGFB3	<Hs.60478>	MGC8471	...	Class
[*, 0.036)	[*, -0.046)	[*, -0.152)	[-0.124, 0.341)	[-0.016, 0.318)	...	Y
[0.036, 0.440)	[0.380, *)	[0.108, *)	[0.341, *)	[0.318, *)	...	Y
[0.440, *)	[0.380, *)	[-0.152, 0.108)	[0.341, *)	[0.318, *)	...	Y
[*, 0.036)	[*, -0.046)	[*, -0.152)	[*, -0.124)	[*, -0.016)	...	N
[0.440, *)	[0.380, *)	[0.108, *)	[-0.124, 0.341)	[0.318, *)	...	Y
[*, 0.036)	[-0.046, 0.380)	[0.108, *)	[-0.124, 0.341)	[-0.016, 0.318)	...	Y
[0.036, 0.440)	[*, -0.046)	[-0.152, 0.108)	[-0.124, 0.341)	[-0.016, 0.318)	...	N
[0.440, *)	[0.380, *)	[0.108, *)	[0.341, *)	[0.318, *)	...	Y
[0.036, 0.440)	[*, -0.046)	[*, -0.152)	Undefined	[*, -0.016)	...	N
Undefined	[-0.046, 0.380)	[*, -0.152)	[*, -0.124)	Undefined	...	N
[*, 0.036)	[*, -0.046)	[0.108, *)	[*, -0.124)	[*, -0.016)	...	N

```
<PMAIP1,[*, 0.036)>      ∧<PC4([-0.716, -0.073)>    → <Class,Y>
<PMAIP1,[*, 0.036)>      ∧<PC4([*, -0.716)>         → <Class,N>
<PMAIP1,[0.036, 0.440)>  ∧<PC4([*, -0.716)>         → <Class,N>
<LTGFB3,[*, -0.152)>     ∧<MGC8471([-0.016, 0.318)> → <Class,Y>
<TGFB3,[0.108, *)>       ∧<MGC8471([-0.016, 0.318)> → <Class,Y>
```

```
<PMAIP1,[*, 0.036)>∧<PC4([-0.716,-0.073)> →<Class,Y>
<PMAIP1,[*, 0.036)>∧ <PC4([*,-0.716)>      →<Class,N>
```

Figure 5.1: *Overview of analysis.* The figure illustrate the analysis that is conducted in this chapter. Compared to the microarray analysis process shown in Figure 3.1, the data mining step has been extended by several substeps. These are feature selection, which selects significant genes; discretization, which transforms the real-valued log_2-ratios in to discrete values; and learning, which finds a rule model.

a satisfactory prediction performance, and generalizing ability. Their accuracy will typically be very low. Finding reducts will also be very expensive in terms of computation time. An exhaustive search is obviously impossible, but even a heuristic search with a genetic algorithm is very time consuming with such a large number of attributes. Therefore, it is necessary to select the genes that have the best ability to discriminate between the classes and only use these genes in the learning step.

After the feature selection step, a decision system is constructed from the chosen genes. The log_2-transformed ratios are, however, real-valued, and a rough set based system can only deal with discrete values. The values of the conditional attributes are therefore discretized. The classifiers are then trained on the discretized data using one of several rule learning algorithms, which are implemented in ROSETTA.

The main results in this chapter are experimental. The utility of the approach is demonstrated by applying it to a set of gastric tumors, which have been examined with microarrays. A total of six different clinical parameters are predicted from this set of microarrays. Moreover, the performance of the feature selection method is studied. The efficacy of several learning and discretization methods implemented in the ROSETTA system are also examined. Their performance is compared to that of linear and quadratic discrimination analysis.

The classifiers are also biologically validated. One of the best classifiers is selected for each clinical parameter, and the connections between the genes used in these classifiers and the parameters are compared to the established knowledge in the biomedical literature.

5.2 Methods

Our learning problem is to predict the class of tumors examined with microarrays. We may formalize this problem as a decision system $\mathcal{A} = \langle U, A \cup \{d\}\rangle$ where universe U is a set of tumors/microarrays. The set A contains a conditional attribute for each gene. The decision attribute d corresponds to a clinical parameter. Each attribute $a \in A \cup \{d\}$ is a function $a : U \to V_a$. V_a contains $\log_2$-transformed ratios for gene a.

In the next section, we present the feature selection method that is used for identification of differentially expressed genes. The discretization and learning methods that are employed in our experiments are described next. We then explain the evaluation methods that are applied for measuring the performance of the classifiers.

5.2.1 Feature selection

The clinical parameters in our study have only two classes each, and we will mainly consider binary classification problems in this section. Let us call the two classes d_0 and d_1 such that $V_d = \{d_0, d_1\}$. Assume that there are n_j objects with class d_j, $(j = 0, 1)$. Let $V_{a,j,1}, \ldots, V_{a,j,n_j}$ denote the values of attribute a for these objects.

ComputePValue:
Input: Values $V_{a,j,1}, \ldots, V_{a,j,n_j}$ for each class d_j $(j = 0, 1)$ and attribute a.
Output: A p-value for attribute a.

```
1: Compute $t(V_{a,0}, V_{a,1}) = \frac{\overline{V}_{a,1} - \overline{V}_{a,0}}{\sqrt{s^2_{a,1}/n_1 + s^2_{a,0}/n_0}}$
2: Move the means of the distributions of $V_{a0}$ and $V_{a1}$ to 0
       • $\widetilde{V}_{a,0,i} = V_{a,0,i} - \overline{V}_{a,0}$, for all $j = 1, \ldots, n_0$
       • $\widetilde{V}_{a,1,i} = V_{a,1,i} - \overline{V}_{a,1}$, for all $i = 1, \ldots, n_1$
3: for $b = 0$ to $B$ do
4:     Draw $n_{a,0}$ values $\widetilde{V}^b_{a,0,1}, \ldots, \widetilde{V}^b_{a,0,n_0}$ with replacement from $\widetilde{V}_{a,0,1}, \ldots, \widetilde{V}_{a,0,n_0}$
5:     Draw $n_{a,1}$ values $\widetilde{V}^b_{a,1,1}, \ldots, \widetilde{V}^b_{a,1,n_1}$ with replacement from $\widetilde{V}_{a,1,1}, \ldots, \widetilde{V}_{a,1,n_1}$
6:     Compute $t(\widetilde{V}^b_{a,0}, \widetilde{V}^b_{a,1})$
7: end for
8: p-value = $\frac{\left|\left\{b \,\middle|\, |t(\widetilde{V}^b_{a,0}, \widetilde{V}^b_{a,1})| > |t(V_{a,0}, V_{a,1})|\right\}\right|}{B}$
```

Algorithm 5.1: *A bootstrap algorithm for computing p-values (from [44, chap. 16.4]).*

The observed sample mean $\overline{V}_{a,j}$ and variance $s^2_{a,j}$ are then

$$\overline{V}_{a,j} = \frac{1}{n_j} \sum_{i=1}^{n_j} V_{a,j,i} \qquad s^2_{a,j} = \frac{1}{n_j - 1} \sum_{i=1}^{n_j} (V_{a,j,i} - \overline{V}_{a,j})^2 \tag{5.1}$$

We select genes/attributes according to their individual ability to discriminate between classes. For each attribute, the mean value for each class is computed, and a statistical test is used to determine if these means are significantly different. We simply test

$$H_0 : \mu_{a,0} = \mu_{a,1} \quad \text{vs.} \quad H_1 : \mu_{a,0} \neq \mu_{a,1}$$

where $\mu_{a,0}$ is the mean of a for the objects labeled with class d_0, and $\mu_{a,1}$ is the mean of a with regard to class d_1. a is selected if the null hypothesis H_0 is rejected. It is well-known that this can be tested using the following t-statistic:

$$t(V_{a,0}, V_{a,1}) = \frac{\overline{V}_{a,1} - \overline{V}_{a,0}}{\sqrt{s^2_{a,1}/n_{a,1} + s^2_{a,0}/n_{a,0}}} \tag{5.2}$$

which has an approximate t-distribution if the measured values are normally distributed (see e.g., [188]). However, the $\log_2$-ratios are often not normally distributed. We therefore use a bootstrap method for computing the significance of each gene.

Bootstrapping [44] is a resampling-based method, which can estimate the distribution and the standard error of a statistic. New data sets are sampled from the original data, and the statistic is computed for each data set. The computed values of the statistic form the estimated distribution.

In our case, bootstrapping is used to estimate the p-value of each attribute. We use the algorithm described in Algorithm 5.1, which is taken from [44, chap. 16]. The t-statistic $t(V_{a,0}, V_{a,1})$ is first computed for the observed values of attribute a. The observed values are then adjusted so that they agree with the null hypothesis. The null hypothesis is $\mu_{a,1} - \mu_{a,0} = 0$, and the estimated distribution of the t-statistic should thus have a mean at 0. However, the mean of the t-statistic computed from the bootstrap samples will be $t(V_{a,0}, V_{a,1})$ as computed from the observed values if the values are not adjusted. The distributions of $V_{a,0}$ and $V_{a,1}$ are therefore shifted to the same mean (which is chosen as 0 in Algorithm 5.1) so that the distribution of difference $\overline{\overline{V}}_{a,1} - \overline{\overline{V}}_{a,0}$ and hence the distribution of $t(\widetilde{V}_{a,0}, \widetilde{V}_{a,1})$ have the mean 0. After adjusting the distributions, B bootstrap sets are created from the adjusted data. Each bootstrap set b is created by drawing n_j values with replacement from the $V_{a,j}$-values for each class j, and the t-statistic is computed from the drawn values. The p-value is simply the number of bootstrap sets that have a t-statistic with a higher absolute value than the observed t-statistic.

We select a set of significant attributes with p-values below a level α. This set may still be quite large compared to the number of objects. Only the significant attributes with the highest t-statistics are therefore used in the final decision system. This is done by sorting the attributes by the t-statistic and selecting the k attributes with the highest t-statistic.

Feature selection for multi-class problems can be treated in the same way. However, the t-statistic is not directly applicable since it expects only two classes. There are nevertheless several different tests that may be used in this case. One option is to test each attribute with a one-way ANOVA test (e.g., [188]). In this case, an f-statistic is used to test the null hypothesis that the classes have the same mean against the hypothesis that at least two classes have different means. An alternative is to test each attribute with the t-statistic, but to test only one class at the time. The pool of the other classes is then considered as the other class so that the t-statistic becomes applicable. The hypothesis test is, in this case, whether the mean of the class is different from the mean of the pool of the other classes. Notice that the last option corresponds quite well to decision rules. These also try to separate one class from the rest of the classes. The t-statistic may thus be preferable even in multi-class problems.

5.2.2 Discretization

Microarray measurements are real numbers and have to be discretized before a learning algorithm is applied to the filtered data set. In this study, we examine the performance of several different discretization methods.

A discretization method finds a set of cuts that divide the range of an attribute a into a set of intervals or bins. Let $C_a = \{c_1^a, \ldots, c_i^a, \ldots, c_n^a\}$ be such a set of cuts for attribute a where $c_1^a < \cdots < c_i^a < \cdots < c_n^a$. The bins are then defined as

$$U_0 = \{x \in U \mid a(x) < c_1^a\} \tag{5.3}$$

$$U_i = \{x \in U \mid c_i^a \leq a(x) < c_{i+1}^a\}, \quad 1 \leq i < n-1 \tag{5.4}$$

$$U_n = \{x \in U \mid c_n^a \leq a(x)\} \tag{5.5}$$

Frequency binning. The first method that we use is frequency binning. This is an unsupervised method that divides the range into n intervals so that the frequency of the objects is the same in each interval, i.e., $|U_i| = |U_j|$ for all $0 \leq i, j \leq n-1$.

Naive discretization. A very simple supervised method sorts the values of an attribute and defines a cut between of two consecutive values if they have different decision classes. We call this method naive discretization. Formally, let $v_1^a < \cdots < v_j^a < \cdots < v_m^a$ be the sorted values of attribute a, and let δ_j^a represent the decision classes of v_i^a, i.e., $\delta_j^a = \{d(x) | a(x) = v_j^a\}$. The C_a is defined as:

$$C_a = \left\{ \frac{v_j^a + v_{j+1}^a}{2} \;\middle|\; \delta_j^a \neq \delta_{j+1}^a \text{ and } \; j = 1, \ldots, (m-1) \right\} \tag{5.6}$$

Entropy-based discretization. The naive method is sensitive to noise and may create too many cuts. Fayyad and Irani's discretization method [49] is more robust. It makes cuts recursively and uses an entropy-based test to determine if a cut should be made. In more details, the entropy of a set of objects U is

$$Ent(U) = -\sum_{i=1}^{l} P(d_i, U) \log_2(P(d_i, U)) \tag{5.7}$$

where $P(d_i, U)$ is the proportion of objects in U that have class d_i, i.e., $P(d_i, U) = \frac{|\{x \in U | d(x) = d_i\}|}{|U|}$, and l is the number of decision classes in U, i.e., $l = |\{d(x) \mid x \in U\}|$.

The entropy gained by making a cut c is

$$Gain(a, c; U) = Ent(U) - \frac{|U_0|}{|U|} Ent(U_0) + \frac{|U_1|}{|U|} Ent(U_1) \tag{5.8}$$

where $U_0 = \{x \in U \mid a(x) < c\}$ and $U_1 = \{x \in U \mid a(x) \geq c\}$. A cut c is accepted if

$$Gain(a, c; U) > \frac{\log_2(|U| - 1)}{|U|} + \frac{\log_2(3^l - 2) - (lEnt(U) - l_0 Ent(U_0) - l_1 Ent(U_1)}{|U|} \tag{5.9}$$

Boolean reasoning/rough set-based discretization. The methods that we have discussed so far discretize only one attribute at time. They may therefore introduce more cuts than are absolutely necessary for discerning between the decision classes. Nguyen and Skowron [121, 120, 122] have introduced a supervised method that considers all of the attributes simultaneously and creates consequently fewer cuts.

Their method is developed with basis in rough sets methods and Boolean reasoning. Let L be the set of pairs of objects with different decision class, i.e., $L = \{\langle x, y\rangle \in U \times U \mid d(x) \neq d(y)\}$, and $v_1^a < \cdots < v_j^a < \cdots < v_{m_a}^a$ be the sorted values of attribute a as before. The method selects a semi-minimal subset P of the set of all possible cuts (over all attributes)

$$C = \left\{ \left\langle a, \frac{v_j^a + v_{j+1}^a}{2} \right\rangle \,\middle|\, a \in A \text{ and } \; j = 1, \ldots, (m_a - 1) \right\} \tag{5.10}$$

such that for any pair $\langle x, y\rangle \in L$ there is some pair $\langle a, c_i^a\rangle \in P$ where c_i^a separates x and y (i.e. $(a(x) < c_i^a < a(y))$ or $(a(y) < c_i^a < a(x))$). P is found with a greedy algorithm that picks cuts iteratively. In each iteration, the cut that discerns between the largest number of object pairs in L is selected and added to P. The pairs that are separated by this cut are then removed from L. The algorithm terminates when L is empty.

Discriminant discretization. As a last alternative we use univariate discriminant analysis for discretization. This is really a classification method in itself. However, it may also be used as discretization method. Basically, we construct a bin for each class. The values of an attribute a are then put into a bin according to the class that would be predicted for the value.

Discriminant analysis is developed with basis in Bayesian decision theory. According to this theory the optimal choice when classifying an object is to assign the object x to class d_1 if

$$p(d_1|a(x)) \geq p(d_0|a(x)) \tag{5.11}$$

where $p(d_i|a(x))$ is the posterior probability. By applying Bayes' theorem

$$p(d_i|a(x)) = \frac{p(a(x)|d_i)p(d_i)}{p(a(x))} \tag{5.12}$$

we may obtain the following inequality from (5.11)

$$p(a(x)|d_1)p(d_1) \geq p(a(x)|d_0)p(d_0) \tag{5.13}$$

If we assume that a(x) is normally distributed given class d_i with mean $\mu_{a,i}$ and variance $\sigma_{a,i}$, i.e.,

$$p(a(x)|d_i) = \frac{1}{\sqrt{2\pi}\sigma_{a,i}} e^{-\frac{(a(x)-\mu_{a,i})^2}{2\sigma_{a,i}^2}} \tag{5.14}$$

and that the variance is the same for each class, we may derive the following allocation rule from (5.13):

Value $a(x)$ should be assigned to the bin U_1 for class d_1 if

$$\frac{\mu_{a,1} - \mu_{a,0}}{\sigma_a^2} a(x) - \frac{\mu_{a,1}^2 - \mu_{a,0}^2}{2\sigma_a^2} \geq v \text{ where } v = \ln \frac{p(d_0)}{p(d_1)} \tag{5.15}$$

and otherwise to U_0. σ_a^2 is the pooled variance. This allocation rule is equivalent to making a cut c_1^a:

$$c_1^a = \frac{\mu_{a,1} + \mu_{a,0}}{2} + \frac{\sigma_a^2}{\mu_{a,1} - \mu_{a,0}} v \tag{5.16}$$

U_1 and U_0 is then defined as $U_1 = \{x \in U \mid a(x) \geq c_1^a\}$ and $U_0 = \{x \in U \mid a(x) < c_1^a\}$. Notice that if the prior distributions are equal, $v = \ln p(d_0)/p(d_1)$ becomes 0, and the cut would be halfway between the two means. This corresponds quite well to the t-statistic (in the feature selection method), which measures the distance between the means of the two classes.

The allocation rule in (5.15) is known as the linear discriminant rule. If we assume that variances of the classes are different, the quadratic discriminant rule follows from (5.13). This rule assign $a(x)$ to the bin U_1 for class c_1 if

$$-\frac{1}{2}\left(\frac{1}{\sigma_{a,1}^2} - \frac{1}{\sigma_{a,0}^2}\right) a(x)^2 + \left(\frac{\mu_{a,1}}{\sigma_{a,1}^2} - \frac{\mu_{a,0}}{\sigma_{a,0}^2}\right) a(x) - \frac{1}{2}\left(\frac{\mu_{a,1}^2}{\sigma_{a,1}^2} - \frac{\mu_{a,0}^2}{\sigma_{a,0}^2} + \ln\frac{\sigma_{a,1}^2}{\sigma_{a,0}^2}\right) \geq v \tag{5.17}$$

We use both the linear and the quadratic discriminant rule for discretization in our study.

5.2.3 Learning algorithms

We train classifiers primarily with 3 different learning algorithms implemented in Rosetta [126]. These methods create the following rule models:

- **Genetic reducts** [193]: This approach finds decision-relative reducts with a genetic algorithm and was originally introduced by Wróblewski [192]. Rosetta has two different versions of this algorithm: Wróblewski's original implementation and an extended reimplementation by Vinterbo and Øhrn [185]. The latter has more parameter options and is used in this study. Both versions can create reducts for a full decision table or on an object-wise basis. In the latter case, reducts and rules are found separately for each object in the decision system. We use the object-wise option in our experiments since this usually gives the best results, in our experience. Vinterbo and Øhrn's implementation also has a facility for specifying indiscernibility definition graphs, and this is used to define a special indiscernibility relation in order to take undefined values into account:

$$IND(A) = \{\langle x, y\rangle \mid \text{for all } a \in A \ (a(x) = a(y)) \text{ or } (a(x) \text{ or } a(y) \text{ is undefined})\}$$

- **Dynamic reducts** [8, 7]: This is a resampling-based method, which selects the most stable decision-relative reducts. A number of decision systems are randomly sampled from the original decision system and decision-relative reducts are computed from these decision systems. The decision-relative reducts that occur most frequently over these systems, are assumed to be more stable, and one may choose to retain only these reducts. Rosetta provides an algorithm, which makes such filtering possible, but the dynamic reduct algorithm in ROSETTA makes no such filtering by itself. Hence, no filtering is used in our study. We compute decision-relative reducts for each sampled decision system with Wróblewski's genetic algorithm.

- **1R classifier** [72]: This method assumes that conditional attributes are independent and creates very simple rules with only one descriptor (attribute-value pair) in the antecedent. A classifier contains one rule for each conditional attribute and each value of the attribute found in the decision system. Each rule has a number of votes, and when an object is classified, the votes from all of the matching rules are added together.

We compare these methods to diagonal linear and quadratic discrimination analysis, which have been used by Dudiot et al. [39] to classify microarray data. An almost identical method was also used by Golub et al. [60] for classification of bone marrow and blood samples taken from acute leukemia patients and analyzed with microarrays.

Diagonal discrimination analysis assumes that there is no covariance between the conditional attributes (such that the covariance matrix is a diagonal matrix). The allocation rule for linear discriminant analysis appears by adding the left-hand side of inequality (5.15) over all conditional attributes so that class d_1 is predicted if

$$\sum_{a \in A} \left(\frac{\mu_{a,1} - \mu_{a,0}}{\sigma_a^2} a(x) - \frac{\mu_{a,1}^2 - \mu_{a,0}^2}{2\sigma_a^2} \right) \geq \ln \frac{p(d_0)}{p(d_1)} \tag{5.18}$$

Class d_0 is predicted otherwise. The allocation rule of quadratic discriminant analysis occurs similarly by adding the left-hand side of inequality (5.17) over all conditional attributes.

5.2.4 Evaluation

The classifiers are tested with leave-one-out cross-validation. The whole development process of the classifier is performed in each iteration of the cross-validation procedure. The features are first selected from the training set and discretized. A classifier is then trained on the resulting data set. Notice that the result would be too optimistic if feature selection was run on the complete data set since the selected features would depend on the test objects in this case (See [157] for a good discussion of this problem).

The performance of the classifiers is estimated with several different measures. In the next section we report accuracy, sensitivity, specificity, and AUC. We state their definition here for the sake of completeness. Let $C(i, j)$ denote the number of objects that have class d_i, but are predicted to d_j. Hence,

$$C(i, j) = \left| \{ x \in U \mid d(x) = d_i \text{ and } \hat{d}(x) = d_j \} \right|$$

where $\hat{d}(x)$ is the class predicted by the classifier. Then accuracy, sensitivity and specificity are defined as:

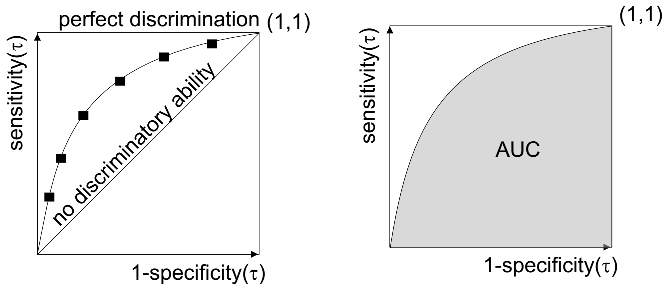


Figure 5.2: ROC and AUC

- $Accuracy = \frac{\sum_{i=0}^{1} C(i,i)}{\sum_{i=0}^{1}\sum_{j=0}^{1} C(i,j)}$

- $Sensitivity = \frac{C(1,1)}{C(1,1)+C(1,0)}$

- $Specificity = \frac{C(0,0)}{C(0,0)+C(0,1)}$

The Receiver Operating Characteristic (ROC) curve [45, 173] is a graphical representation of the discriminatory ability of the classifier. Many learning algorithms (e.g., neural networks) do not predict a class for an object directly, but output its degree of certainty that the object belongs to class d_1 instead. Class d_1 is then predicted if the certainty is greater or equal to a threshold τ. The ROC is a plot of $sensitivity(\tau)$ vs. $(1 - specificity(\tau))$ for varying values of this τ ($\tau \in [0, 1]$). An illustration is given in Figure 5.2. Rosetta computes its certainty by the voting scheme given in Section 4.7.

The area under the ROC curve (AUC) is a measure of the performance of the classifier. An important feature of the ROC curve and the AUC is that they are independent of the class distributions and costs associated with incorrect predictions. The accuracy, on the other hand, assumes that the cost of an incorrect prediction is the same for all classes and that the classifier will be applied on instances with a class distribution identical to the distribution in the testing set.

5.3 Data and preprocessing

Our method was applied to a data set of gastric carcinomas. Gastric cancer is the second most frequent cause of cancer related deaths on a world-wide basis. The disease is often detected at an advanced stage, and 5-year survival rate is between 10–20%. There are presently no known genes that allow classification of gastric carcinoma with respect to important clinicopathological parameters. Molecular classification of gastric tumors is also important for developing new diagnostic and treatment procedures.

Clinical parameter	***Classes (Distribution)***
Laurén's histopathological classification	Diffuse (8) or Intestinal (9)
Localization of tumor	Cardia (4) or Non-cardia (13)
Lymph node metastasis	Yes (10) or No (7)
Penetration of the stomach wall	Yes (13) or No (4)
Remote metastasis	Yes (3) or No (10)
Serum gastrin	High (5) or Normal (9)

Table 5.1: *The clinical parameters for which classifiers were built.* The number of objects for each class is shown in parentheses.

5.3.1 Data material

A total of 17 gastric carcinomas were examined with microarrays. One microarray was used for each tumor. Each array contained probes for 2,504 human genes, and each probe was printed twice on each array. An mRNA mixture of 10 different cell lines was used as reference on all microarrays. The reference sample was labeled with the fluorescent dye Cy3, and the tumor samples were labeled with Cy5.

We developed classifiers for the clinical parameters shown in Table 5.1. Several of these parameters are directly related to the extension of the disease. Penetration of the stomach wall describes whether a tumor has penetrated to the external surface of the stomach. Lymph node metastasis indicates whether a tumor has spread to local lymph nodes, and remote metastasis denotes whether a tumor has spread to remote organs. The other parameters examine the biological features of the tumor. Laurén's histopathological classification system describes the growth pattern of a tumor. This system is based on microscopical inspection of tumors and correlates to patient characteristics like age, sex, etc. Gastrin is a hormone-like substance that stimulates gastric mucosal growth and can be measured in the serum of the patient. The localization describes the site in stomach where the primary tumor is found. The stomach is divided into 3 parts: cardia (the uppermost), corpus, and antral (the lowest). The cardiac tumors are epidemiologically distinct from tumors in other parts. Hence, we distinguish between cardia and non-cardia in this study.

5.3.2 Preprocessing of microarray measurements

The microarrays were scanned with a confocal laser scanner at two different wavelengths – one for each dye. The images produced by the scanner were analyzed using Scanalytics' MicroArray Suite, and globally normalized to balance the intensity between the two dyes.

Further analysis was done on the $\log_2$-transformed ratios. Scatterplots of the ratios revealed a fish-tail-like distribution, indicating higher variance of $\log_2$-ratios for spots with low average intensity. Unreliable spots were thus removed. Filtering was done individually for each array by selecting one of the following filtering criteria:

1. A spot was retained if the signal was greater than the corresponding background signal for both dyes.

2. A spot was retained if the signal was greater than two times the corresponding background signal for both dyes.

3. A spot was retained if the ratio quality of the spot reported by MicroArray Suite was above a certain threshold.

The criterion for each array was determined by applying all criteria in turn to the array. The criterion that retained the most spots and had a correlation coefficient between the retained duplicate spots of at least 0.70, was chosen and used for filtering of that array.

Since there were two spots for each gene probe on the microarrays, the $\log_2$-ratios from these duplicates were averaged. A gene was considered undetectable and was removed, if the ratios for both spots were missing on more than 50% of the microarrays. This removal of undetectable genes was applied during cross-validation such that the number of undetectable genes varied slightly for each iteration of this procedure.

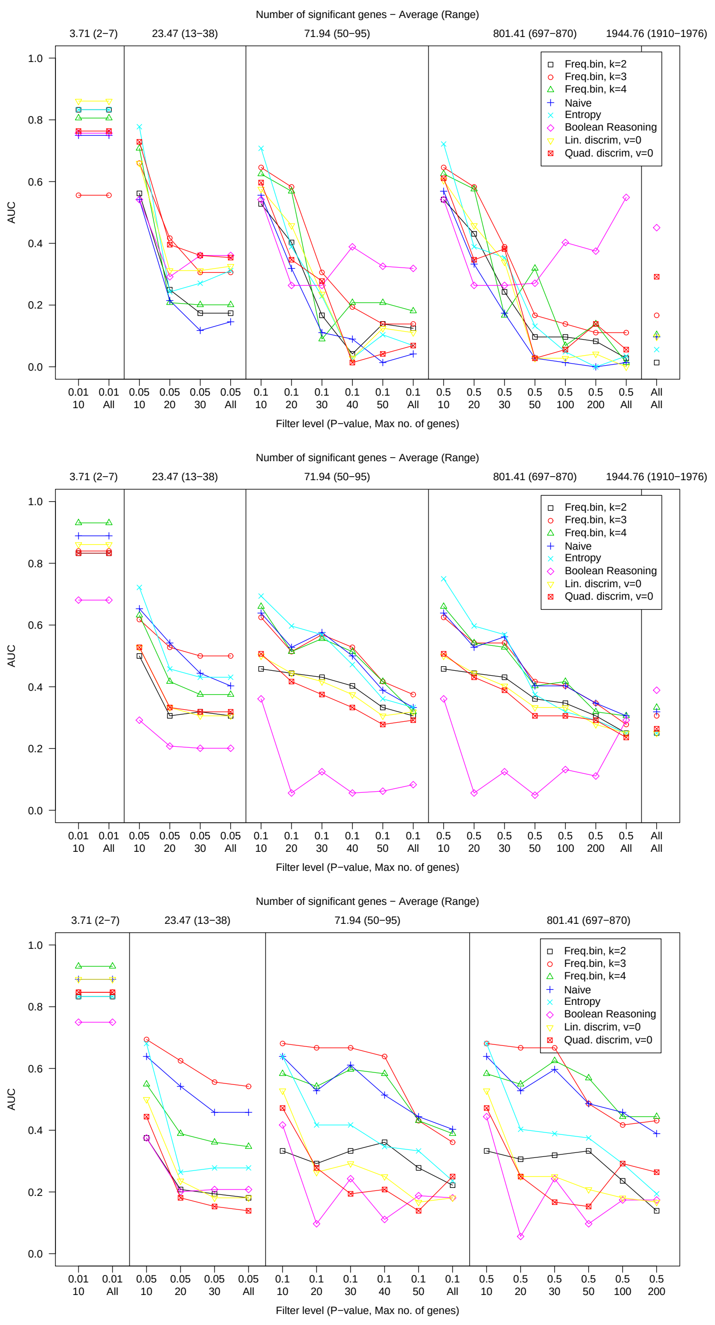
5.4 Results

We examined the classification performance for each of the six clinical parameters. In particular, the dependence of the classifiers on the feature selection method was determined. Moreover, we also sought to identify if some discretization or learning method was better suited for this task. Finally, the best classifiers are then inspected, and the genes used in them are compared to the established knowledge in the biomedical literature. The results are reported in the following section.

5.4.1 The feature selection

We tried to build classifiers without any selection at first. However, the results were quite poor. The AUC and accuracy were often well below 0.5. Hence, the classifiers performed worse than random guesswork. The running time was quite long. It took about 2.5 CPU hours on average to run leave-out-one cross-validation for the genetic reduct algorithm on Laurén's histopathological classification system using a Sun Server with four 750 MHz UltraSPARC III CPUs and 8 GB internal memory. Leave-out-one cross-validation for the 1R Classifier took about 1 CPU hour on average. The dynamic reduct algorithm was tried only once on the full data set and used over 8 days to finish. Further experiments with this algorithm were consequently made on smaller data sets. These usually contained 200 attributes or less since the time consumption of the dynamic reduct algorithm was quite high for decision systems with more attributes.

The first row of numbers on the horizontal axis displays the significance level and the second number gives the maximum numbers of genes that were allowed in the classifiers. The number of genes that were significant at a particular level is shown on the horizontal axis above each plot. The initial filtering of undetectable spots was applied during cross-validation so that the number of genes varied even when the significance level was set to 1 and all genes were admitted.

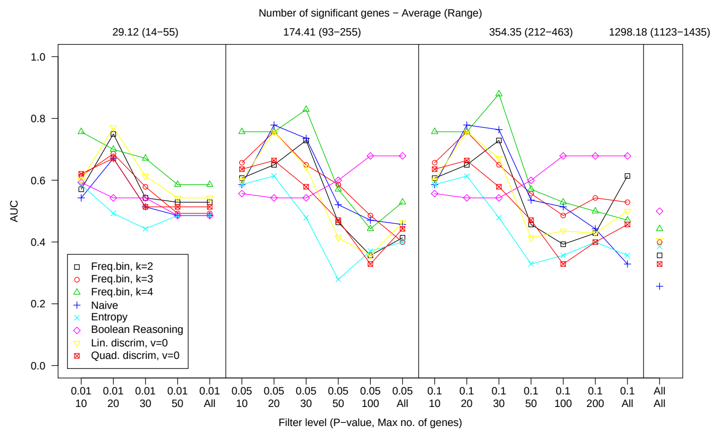


(a) Genetic reducts

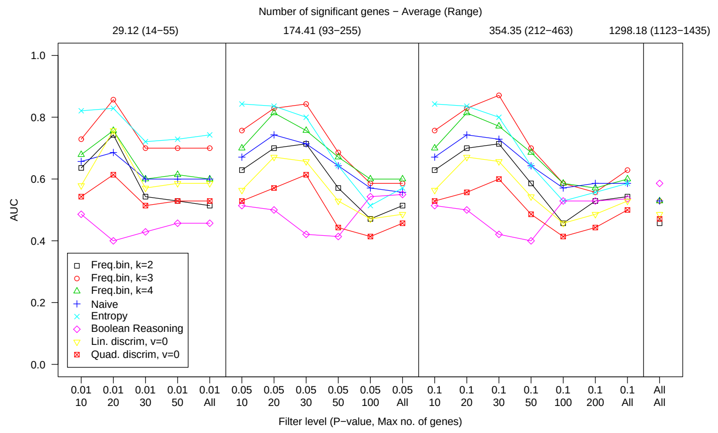
(b) 1R Classifier

(c) Dynamic reducts

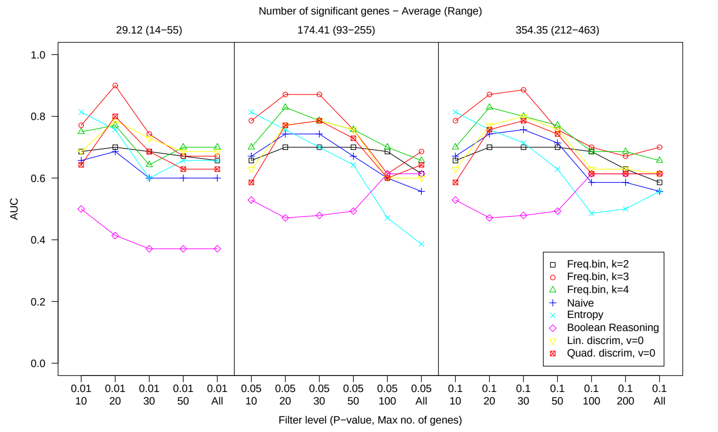
Figure 5.3: *AUC for Laurén's classification system over different filtering levels.*



(a) Genetic reducts

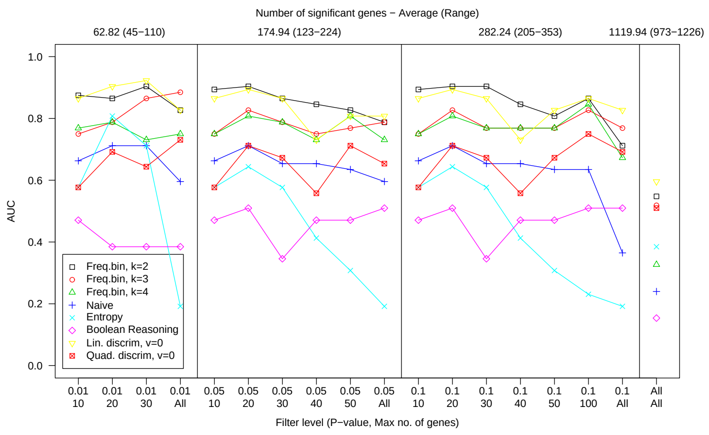


(b) 1R Classifier

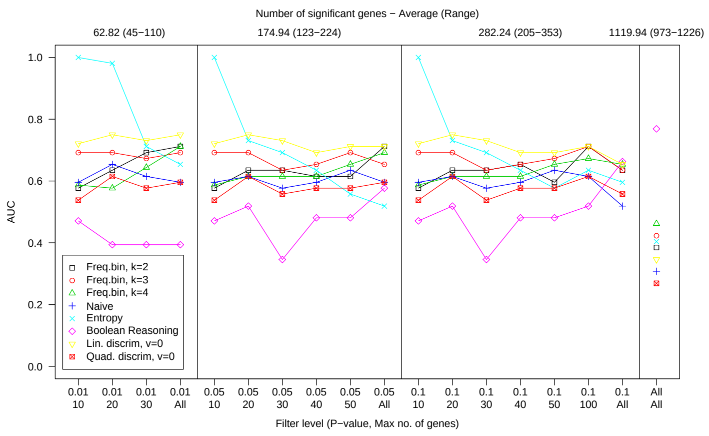


(c) Dynamic reducts

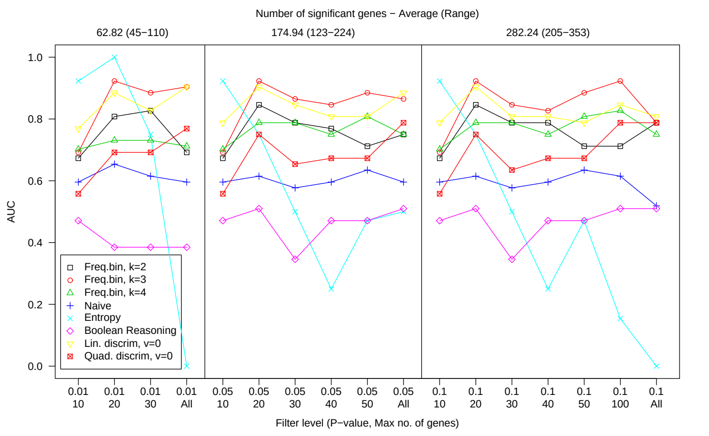
Figure 5.4: *AUC for lymph node metastasis over different filtering levels.*



(a) Genetic reducts

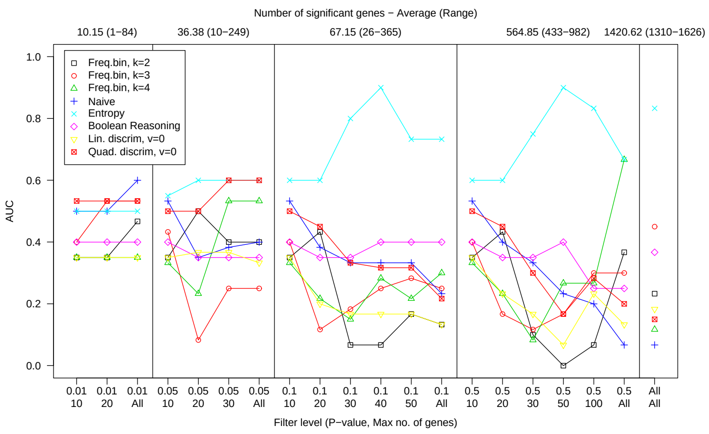


(b) 1R Classifier

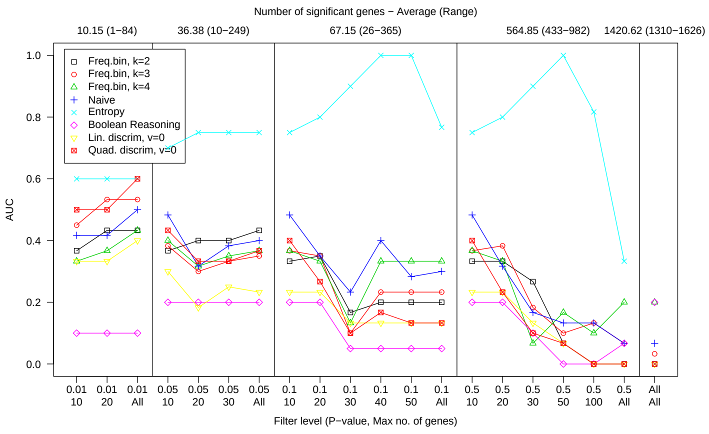


(c) Dynamic reducts

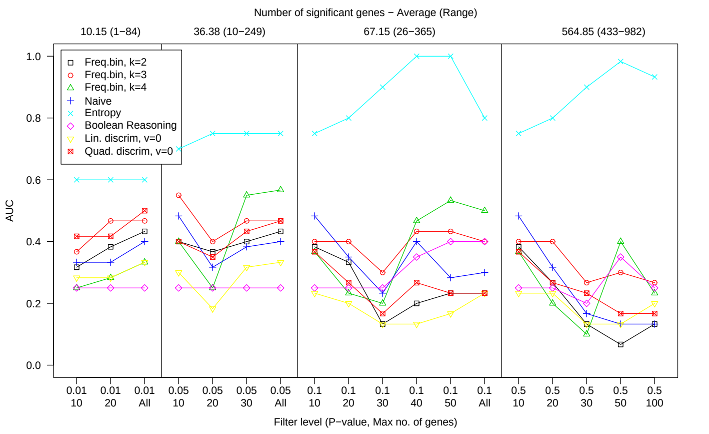
Figure 5.5: *AUC for localization of the tumor over different filtering levels.*



(a) Genetic reducts



(b) 1R Classifier



(c) Dynamic reducts

Figure 5.6: *AUC for remote metastasis over different filtering levels.*

We examined the performance of the discretization methods and the learning algorithms on the Laurén parameter. The results are shown in Figure 5.3. The AUC clearly decreased with increasing number of attributes. The best AUC was achieved at significance level 0.01. At this level there were less than 10 significant attributes. We did not achieve similar AUC-values at higher significance levels (where genes with higher p-values were admitted), but found that the performance decreased when the maximum number of attributes, which were permitted in the classifiers, was increased. Nearly all combinations of the discretization and learning methods displayed a similar behavior over different feature selection levels. In particular, the performance shown by various discretization methods was very similar when the genetic reduct algorithm or the 1R classifier was used. One exception was the Boolean reasoning discretization. It had a more a U-shaped performance. Its AUC would first decrease to a minimum and then increase when more attributes were permitted.

The results for lymph node metastasis, which are shown in Figure 5.4 were fairly similar. The AUC fell with increasing number of attributes for most methods. However, more attributes were significant at the 0.01 level, and the best performance was achieved with 20 attributes. The AUC decreased with more attributes, and it was necessary to restrict the number of attributes in this case. Similar behavior was seen at higher significance levels. Boolean reasoning discretization again showed a different behavior than the rest. Its AUC performance was U-shaped or just increased as a function of the number of attributes.

The training data had a skew class distribution for several of the parameters. This included penetration of the gastric wall, localization of primary tumor, remote metastasis, and serum gastrin as shown in Table 5.1. There were markedly fewer objects for one of the classes. The classifiers that were trained for the parameters had a tendency to predict only the majority class, and the performance depended to a large extent on the discretization method that was used. Feature selection also had less influence on some of the parameters.

One example is the classifiers that predicted the localization of primary tumor as illustrated in Figure 5.5. The AUC value was clearly lower when all of the genes were used, but there is no apparent trend showing a decreasing performance with increasing significance level and increasing number of attributes as for Laurén and lymph node metastasis. The only exception was the entropy-based discretization method, which was very sensitive to the filtering level, but also yielded some of the best results.

For remote metastasis, there was some decrease in the AUC with an increasing number of genes for most of the discretization methods (as shown in Figure 5.6). However, the AUC value was mostly below 0.5 so that overall performance was quite bad. Satisfactory results were obtained with entropy-based discretization, but performance of this method was odd and unexpected. The AUC value increased until 40–50 genes were used at 0.1 significance level. This is a quite high significance level, and one would assume that the genes with lower p-values would be better at discriminating between the classes. One explanation may be that we had very little data for this parameter. It was known for only 13 of the tumors whether they had remote metastasis, and only 3 tumors belonged to the minority class (Yes).

The results for penetration of the stomach and serum gastrin were quite similar to remote metastasis. However, the entropy-based method behaved to a larger extent as expected and the AUC dropped when number of genes was increased. Only this discretization method gave interesting results. The rest of the discretization methods had AUC below 0.5.

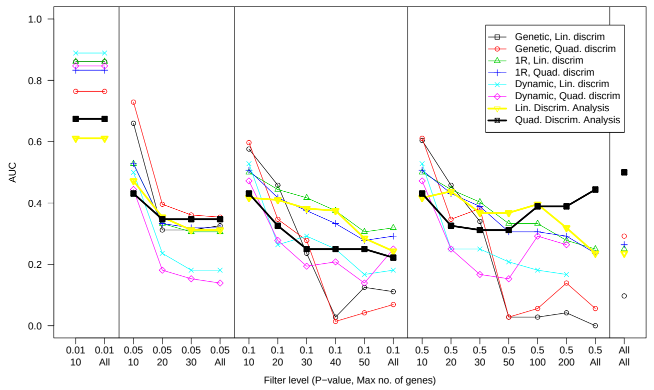
5.4.2 The discretization methods

The discretization methods behaved clearly differently for the clinical parameters with even class distribution and the parameters with skew class distribution. For Laurén and lymph node metastasis, the different discretization methods (except for Boolean reasoning) gave quite similar results, and their AUCs were correlated to the significance level and maximum number of genes permitted in the classifiers. Given the small sample size, we have to be cautious when comparing the methods. Still, some methods seemed to give better results than others. Good results were obtained with frequency binning, but this depended on the number of bins. Three and four bins usually worked better than only two bins, which produced some of the worst results. The entropy-based and the naive method sometimes performed well. Linear discriminant discretization worked quite well with the genetic reduct algorithm, but otherwise the results were not so good. Quadratic discriminant discretization performed worse than the linear discriminant method and usually had one of the worst AUC values. Boolean reasoning discretization had also a poor performance, and it had a tendency to increase with increasing number of attributes.

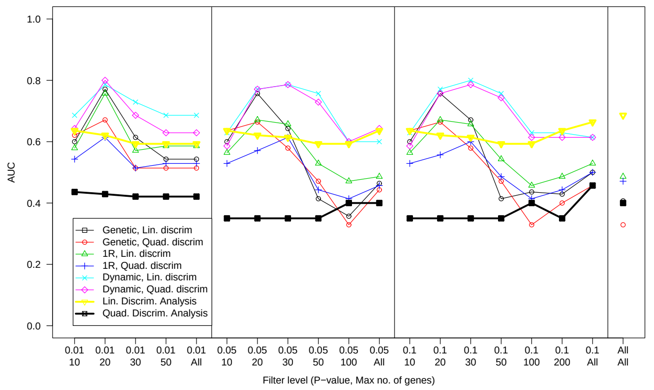
For the parameters with skew class distribution, entropy-based discretization produced the best results. The AUC obtained with other methods was mostly unsatisfactory for penetration, remote metastasis, and serum gastrin. The classifiers for localization behaved more like those of Laurén and lymph node metastasis. Most discretization methods performed well. Entropy-based discretization gave the best results for the 1R classifier and dynamic reducts, but even for this parameter the method behaved quite differently from the rest. Linear discrimination and frequency binning gave good results, while quadratic discrimination and naive discretization were not very effective. Boolean reasoning discretization had the worst performance.

One might question the learnability of penetration, remote metastasis, and serum gastrin, since only entropy-based discretization gave satisfactory results. In particular, the results of remote metastasis and serum gastrin seem strange and weak. Localization on the other hand seems more reliable, since several methods gave good results. It might be that the performance of the entropy-based method is just an artifact of the data, and the classes cannot really be separated using the available microarray data. However, since this performance was systematic over four different parameters, it appears to have some merit, and we conclude that the entropy-based method is better for discerning between classes with skew distribution.

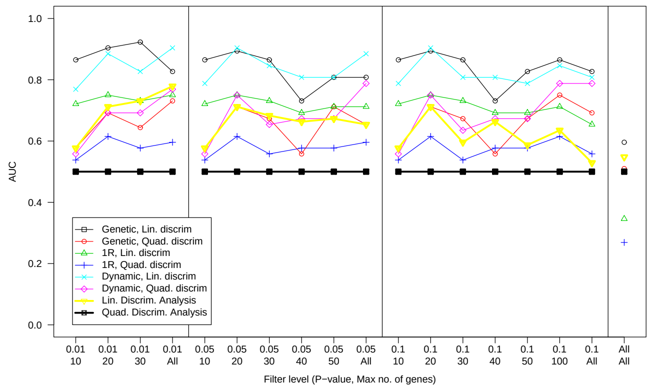
We compared the performance of discriminant discretization to discriminant analysis. The AUC of each rough set based learning method combined with discriminant discretization was plotted together with AUC of discriminant analysis methods for



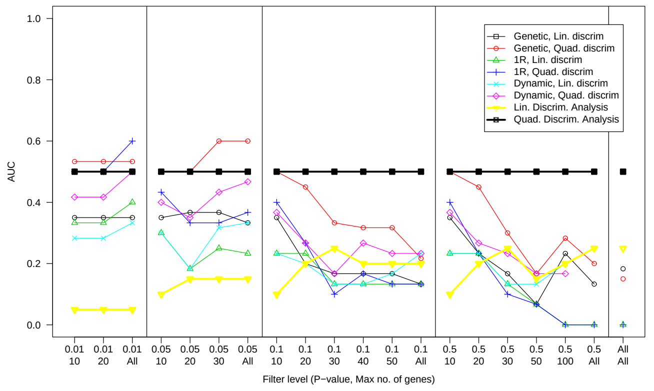
(a) Laurén



(b) Lymph node metastasis



(c) Localization



(d) Remote metastasis

Figure 5.7: *Comparison of the discrimination-based discretization and discrimination analysis.*

4 of the parameters. This comparison can be found in Figure 5.7. The results for penetration and serum gastrin were similar to the results of remote metastasis. Linear discrimination usually gave better results than quadratic discrimination. This held both when these methods were used for discretization and when they were applied for classification. More importantly, the rough set based learning methods had usually better AUC than discrimination analysis, but were more sensitive to the filtering level. This was particularly evident for Laurén, lymph node metastasis and localization. The results for penetration, remote metastasis, and serum gastrin were more complex, and discriminant analysis, in particular quadratic discrimination analysis, had often higher AUC than the rough set based methods. However, the results for these parameters were not satisfactory for any of the methods. The AUC was mostly lower or equal to 0.5. Hence, it seems that better results could be obtained using discriminant discretization together with one of the rough set based learning algorithms than with discrimination analysis alone.

Discriminant discretization was tested with different values for v. We found that setting v to 0 (equal class distribution) gave better results than setting this variable according to the class distribution of the attribute after removal of missing values. However, this did not have any effects for the discriminant analysis.

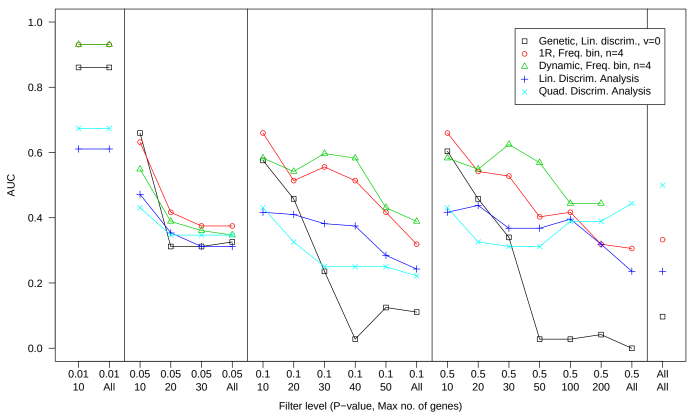
5.4.3 The learning methods

We compared the performance of the learning algorithm by plotting the AUC-value of each algorithm over different filtering levels. For the algorithms that depended on discretization, we selected the discretization method that gave highest AUC-value. The results are given in Figures 5.8 and 5.9.

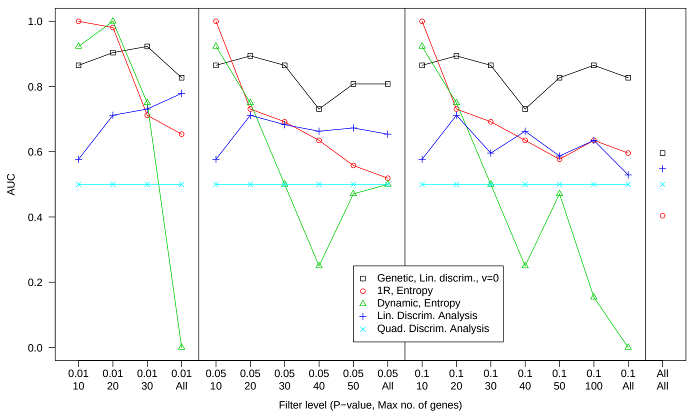
The performance of the 1R classifier and the dynamic reduct algorithm were quite similar. They gave the best results for the first four parameters, but the genetic reduct algorithm achieved the highest AUC value for penetration and serum gastrin. Thus, there is no strong evidence that any of these methods may be better than the others for learning classifiers from the microarray data. However, the 1R classifier and dynamic reduct algorithm performed better than the genetic algorithm for more parameters. The best results for Laurén and lymph node metastasis – the parameters with even class distribution – were also obtained with these two algorithms. Therefore, there is some indication that these two methods worked better than the genetic reduct algorithm

One would expect the 1R classifier to work well since the feature selection method picks attributes individually according to their ability to discriminate between the classes. Each of the selected attributes may therefore be used separately as a classifier. Moreover, it will often be advantageous to use several or all of the selected attributes in a classifier such that a prediction is based on several attributes, since the values for some attributes may be distorted by noise or may be missing.

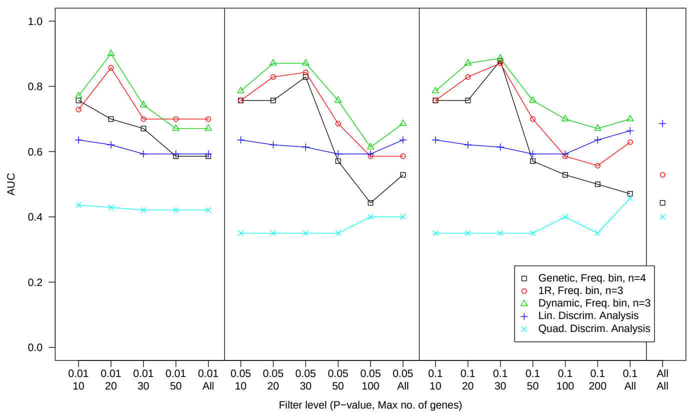
This is exactly what the 1R classifier does. It assumes that the attributes are independent and makes rules separately for each attribute selected in the feature selection phase. When a prediction is made, one of the rules for each attribute will



(a) Laurén

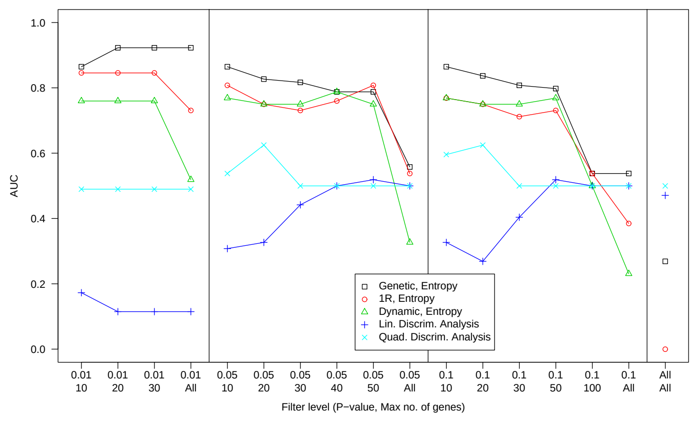


(b) Localization

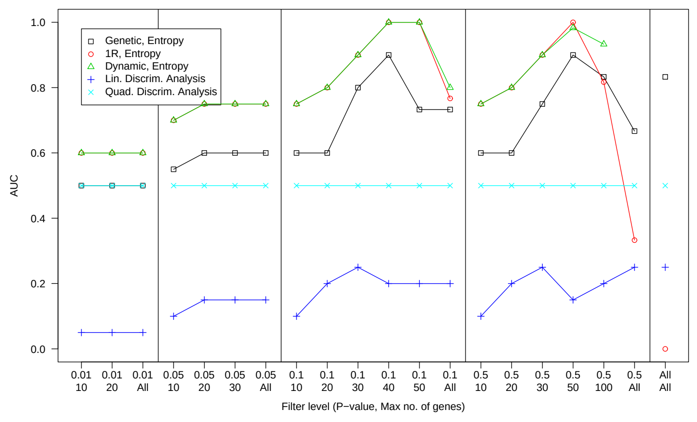


(c) Lymph node metastasis

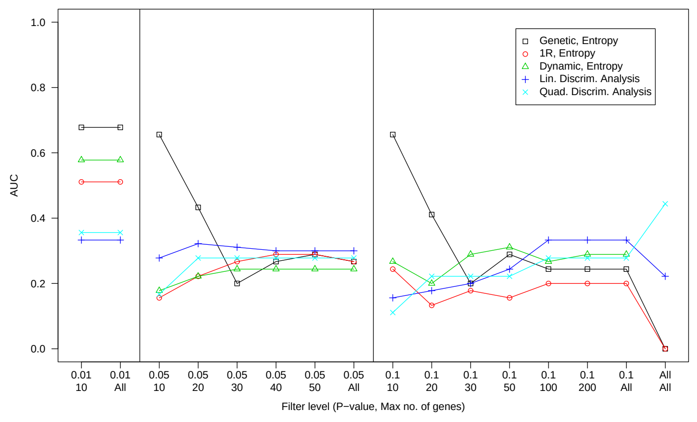
Figure 5.8: *The performance of the each learning method using the discretization method that achieved the best AUC-value.*



(d) Penetration



(e) Remote metastasis



(f) Serum gastrin

Figure 5.9: *The performance of the each learning method using the discretization method that achieved the best AUC-value.*

match the object, and the votes associated with these rules are added. A single attribute has little significance on the result. What matters is the total contribution of all of the attributes. Hence, it will be less affected by noise in some of the attributes.

The genetic reduct algorithm tries to find combinations of attributes that discern well between the classes and makes another selection from the already chosen attributes. However, it is not certain that using combinations of the already selected genes will give a better result than using all of them. Since each attribute selected by the feature selection method is, in principle, a reduct, the reducts found by the genetic reduct algorithm will not contain many attributes. Moreover, only a limited number of reducts is found so that the classifier will rely on fewer attributes and be more vulnerable to noise. One would expect that the performance of genetic reduct algorithm was less sensitive to the number of attributes admitted by feature selection so that its performance should decrease less than the performance of the 1R classifier when this number was increased. We observed such behavior sometimes, but the AUC value usually dropped just like in the other methods. The algorithm may work better than 1R classifier when class distribution is skewed. For such parameters, it may be more difficult to define good cuts during discretization, and combinations of attributes may be more useful. The algorithm worked quite well in fact for localization, penetration, and serum gastrin.

The dynamic reduct algorithm also makes a selection, and it uses the genetic reduct algorithm as a subprocedure. However, it creates more reducts by resampling and achieves a similar effect to the 1R classifier in this way. A classifier built with this method relies on many more rules and genes when it makes a prediction. Unfortunately, it has a long running time.

An interesting result is the performance of the linear and the quadratic discriminant methods compared to the other methods. Both methods usually had a lower AUC-value than the rough set based methods. In particular, the quadratic method performed poorly. However, these methods are quite similar to the 1R classifier. All of them assume that the genes are independent such that they should perform fairly similarly. However, they did not, and the reason is probably that discrimination analysis assumes that the measurements of each gene are normally distributed.

5.4.4 Biological validation of the best classifiers

We selected one of the best classifiers of each clinical parameter for further examination. The details of these classifiers are shown in Table 5.2. Most of these classifiers had a very good accuracy and a high AUC value. Attributes that were significant at the 0.01 level were primarily chosen, and at most 10 or 20 attributes were allowed in the classifiers. We had to relax the significance level to 0.05 in order to get satisfactory accuracy for serum gastrin. For remote metastasis, a significance level of 0.10 and a maximum of 40 attributes were used. Figure 5.10 displays the AUC of all learning and discretization methods at the filter level chosen for each parameter.

The rules and genes used by the classifiers made during leave-one-out cross-validation were collected for each parameter so that they could be inspected by biologists.

Sample	*Method*	*Discretization*	*Fallback*[⋆]	k[†]	α[‡]	*Thres.*
Laurén	Dynamic	Freq. bin (n=4)	Intest. (0.6)	10	0.01	0.458
Localization	Dynamic	Entropy	Cardia (0.6)	20	0.01	0.425
Lymph node meta.	Dynamic	Freq. bin (n=3)	Yes (0.6)	20	0.01	0.655
Penetration	1R	Entropy	No (1.0)	20	0.01	0.188
Remote metastasis	1R	Entropy	Yes (1.0)	40	0.10	0.625
Serum Gastrin	Genetic	Entropy	High (1.0)	10	0.05	0.125

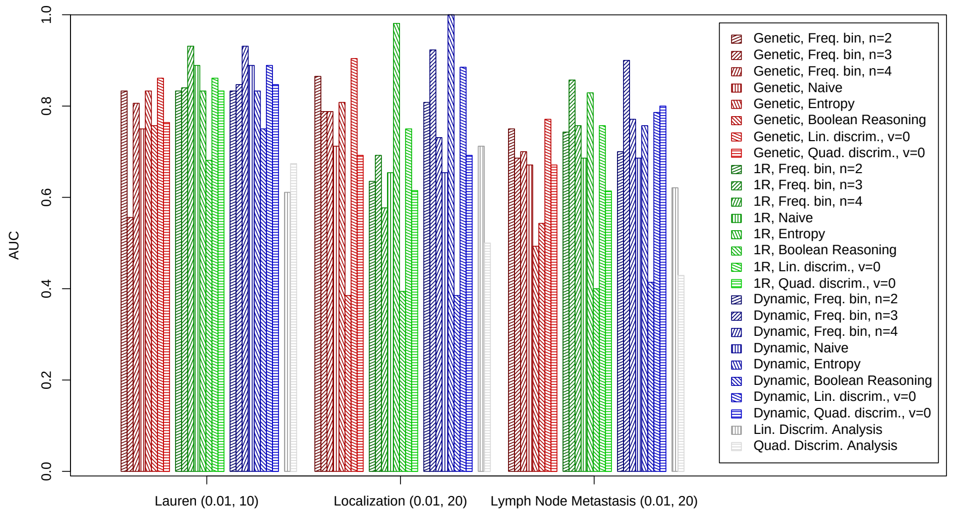
[⋆]The class that was predicted if no rule applied. The certainty ascribed to this class is given in parentheses. [†]Maximum number of genes allowed in a classifier. [‡]Significance level.

Sample	*Accuracy*	*Sens.*	*Spec.*	*AUC*	*No. of rules* *Average*	*Range*	*No. of genes*
Laurén	16/17=0.94	1	0.86	0.93	24.1	10–67	17
Localization	17/17=1	1	1	1	238.1	200–311	72
Lymph node meta.	14/17=0.82	0.7	1	0.9	388.1	222–523	73
Penetration	16/17=0.94	1	0.75	0.85	109.6	28–280	75
Remote metastasis	13/13=1	1	1	1	425.1	305–468	161
Serum Gastrin	11/14=0.79	0.9	0.6	0.66	47.9	18–72	42

Table 5.2: *Summary of the performance of the best classifiers.* The first table contains the parameter setting for the chosen classifiers. The second contains the results for these classifiers. This table is divided into two parts where first part displays the estimated performance while second part reports the number of rules and genes used in the best classifiers. The first two columns display the average number of rules used in each classifier, and the range. The last column displays the total number of genes collected from the classifiers for each parameter.

The range and average of number of rules used in each of these classifiers are reported in the last part of Table 5.2. This table also displays the total number of genes that appear by collecting the genes from all of the classifiers of a parameter. As shown, the number of rules varied from an average of 24 rules for Laurén to 425 rules for remote metastasis. The number of genes varied between 17 and 161.

These numbers obviously depended on the number of genes selected in the feature selection phase as well as the learning algorithm. The genetic reduct algorithm makes its own selection from the already selected genes, and builds rules from the selected genes. The 1R classifier makes no such selection and uses all of the genes admitted by the feature selection. Consequently, it creates more rules and uses more genes. The dynamic reduct algorithm makes more reducts by resampling and has a performance that is more similar to the 1R classifier.



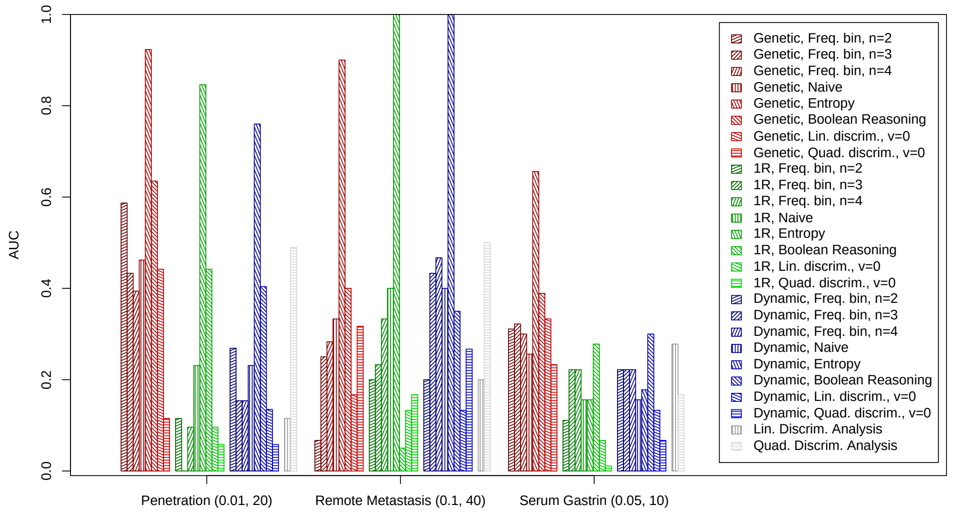


Figure 5.10: *Results for each of the classifiers given the best filtering level.*

Most of the collected genes were used in the classifiers of only one parameter, but some genes were not specific to one parameter and occurred in the classifiers of several different parameters. However, even these genes seemed to be more strongly related to one parameter since they were used frequently for only one parameter. Hence, we assumed that the occasional use of some genes was due to noise, and that these genes were only related to the parameter for which they occurred frequently. In particular, genes that were used only once for a parameter, were ignored.

Six genes, however, occurred frequently for both lymph node metastasis and localization of tumor such that we could not assign them to only one of the parameters. The reason seems to be that these parameters were partially confounded. All cardiac tumors had lymph node metastases, and all tumors without lymph node metastases were thus non-cardiac (Some tumors with lymph node metastases were also non-cardiac so that the parameters were not totally confounded). Further study is required to indicate whether these genes are useful for classification of both parameters and whether their occurrence in many classifiers for both parameters is associated with this specific collection of tumor material.

The biomedical literature was examined in order to investigate whether it was possible to verify the connection between a gene and a parameter detected by our learning method by using established knowledge. However, the literature is not very comprehensive (which is one of the reasons for doing this study in the first place). For many of the genes, there is presently no information available at all, and for most of the genes, it was not possible to find any known association with the clinical parameters or with gastric cancer or other kinds of cancer in general. For some genes, literature search revealed knowledge confirming the connection between the gene and the parameter. Table 5.3 provides a summary of the known connections where we have divided the confirmations into 4 groups according to the kind of evidence that was found. A known connection to the parameter in gastric cancer provides the strongest confirmation between our findings and knowledge present in the literature, but a similar connection found in another type of cancer is also a strong confirmation that this particular gene may play a role in this clinical parameter. A connection only to either gastric or another type of cancer is obviously weaker. We found no information directly contradicting our indications that the genes could be related to the clinical parameters or to cancer biology in general. A detailed account and discussion of these genes is given in Nørsett et al. [124].

5.5 Conclusions

Classification of tumor microarray data is an important biomedical problem that raises thorny computational modeling issues. A classifier built from microarray data may provide an early detection of cancer. Such a classifier may also allow determining tumor subgroup for an incident of cancer. This means that the treatment may be made more effective since it may be started earlier and may be designed specifically for the tumor subgroup. Moreover, classifiers built from microarrays may provide

Parameter	***Known connection to the parameter in***		***Known connection to***		***Unknown connection***
	gastric cancer	***other cancer***	***gastric cancer***	***other cancer***	
Laurén				1	2
Localization	2		3		22
Lymph node metastasis	1	2	1		26
Penetration		4	1	1	17
Remote metastasis	3	2			47
Serum gastrin			1	1	18

Table 5.3: *Confirmation of the involvement of genes used in the gastric cancer classifiers by information found in the biomedical literature.* The table displays the number for each parameter according to the kind of confirmation that was found. It was not possible to find confirmation for many of the genes, and the last column gives the number of genes that were used by more than one classifier, but could not be confirmed in the literature to have any known association with gastric cancer or other cancer.

insight into the underlying biology of tumors. By inspecting such classifiers, we may find genes that may have a significant impact on the differentiation of the tumors into subgroups or on other clinical parameters.

We have shown that rough set methods are applicable in classification of tumors from microarray data. However, feature selection is necessary in order to obtain satisfactory results since a microarray data set has typically very many attributes and few objects. Our approach where bootstrapping is used for feature selection and rough set methods are used for learning is quite general, and it may be used in any classification study of microarray samples.

The feature selection method given in Section 5.2.1 chooses genes by measuring their individual discriminatory ability. A gene need not discern equally well between all objects. For example, one gene may discern well between some objects, while another gene may be better at discerning some other objects. Groups of the selected genes may consequently yield a better discriminatory ability than each gene alone. Hence, the classifiers in this study use all or subsets of the selected genes.

However, genes whose discriminatory ability is only apparent when considered in combination with other genes will not be selected. For example, two genes can be associated with two classes, e.g., d_0 and d_1, in the following manner: If one of the genes is up-regulated and the other is down-regulated, the tumor sample has class d_1. If either both genes are down-regulated or both are up-regulated, the sample has class d_0. In this case, the classes must be determined through an exclusive-or of the expression of the genes. This means that the genes will not be individually significant and will not be selected with the feature selection method in Section 5.2.1. The approach may therefore fail to built a classifier.

We may, however, discover such combinations directly with the genetic or the dynamic reduct algorithms or with a feature subset selection method. Unfortunately, this requires a large number of samples. The number of tumors in our data set is much too small for making such discoveries.

Some of the discretization methods seemed to work better than others. Frequency binning and entropy-based discretization gave good results. Discretization based on linear discriminant analysis was also useful. The entropy-based method appeared to handle skewed class distributions better than the other methods.

Boolean reasoning discretization often had poor performance and behaved differently from the rest of the discretization methods. The AUC had a tendency to increase with additional genes. It is likely that this is due to the global nature of this method. The method considers all attributes at once when it creates cuts. The feature selection method, on the other hand, selects genes individually such that each selected gene may be a good classifier in itself. This means that it is more appropriate to make cuts individually for each gene in this case. The Boolean reasoning approach is consequently less suited for this problem, but it may yield a good performance in other situations (see [119] for some examples).

The rough set based learning methods worked quite well and out-performed both linear and quadratic discriminant analysis. However, there is no strong evidence that one of the rough set based methods was better than any of the other rough set based methods. The performance of the 1R classifier and the dynamic reducts algorithm were very similar, and one may prefer the 1R classifier over the dynamic reducts algorithm since the running time of the former is much shorter.

We obtained high quality classifiers for at least 2–3 of the clinical parameters in our study. The quality of the classifiers for penetration, serum gastrin, and remote metastasis is perhaps more questionable, but the classifiers obtained with entropy-based discretization for these parameters had a plausible performance.

The genes used in these classifiers were examined, and their connection to the parameters was confirmed for several of the genes. However, most of the associations between the genes and the parameters could not be verified by the biomedical literature. For these genes, further experimental work is needed to determine if there is a biological connection between the genes and the respective parameters. This finding demonstrates that the status of present biomedical knowledge is still very scarce and fragmented. Our classification results provide valuable hypotheses regarding which genes should be tested for their possible involvement in gastric cancer.

The sample size in our study was quite small. Several of the clinical parameters had a very skewed class distribution where as few as 3–4 objects belonged to the minority class. Obviously, it is quite difficult to learn from that few objects. There is a risk that the classification performance was due to artifacts in the data set. One such indication is that the performance of classifiers for these clinical parameters depended more on the discretization and learning method. More reliable results can be obtained by increasing the samples size.

Furthermore, our study included only $2,504$ genes out of a total of at least $30,000$ genes in the human genome. Some of the genes that were not included in our study may have a connection to the parameters. A further study should not only include more tumors, but also more genes.

Still, our results show that it is possible to develop classifiers with a small number of tumor samples, and that rough set based methods may be well suited for this task. We believe that rough set based learning combined with feature selection may become an important tool for classification of microarray samples.

Learning Annotations in a Gene Ontology

6

6.1 Introduction

Sections 3.5.1 and 3.6.1 discussed the use of microarrays in prediction of gene function. In this kind of study, some cells are exposed to an external stimulus, and the expression response in the genes is assessed with microarrays. The response of each gene can be measured as a time sequence that describes the behavior over a period of time. Co-expressed genes that have a similar time response are then considered to have the same function. The underlying assumption for making such conjectures is that their expression profiles are similar since they are regulated by the same process.

The function of a gene may be predicted by a supervised learning algorithm with a basis in this assumption. However, such an algorithm needs a set of (unrelated) classes that may be predicted. Sometimes it is obvious what classes may be used for this purpose, but for gene function this is not so evident. The classes have to be defined.

At the moment, there exist several ontologies that define a set of classes for gene function. Two of these (the MIPS[1] ontology [57] and the Gene Ontology [29]) were mentioned in Section 3.6.1. The classes in these ontologies are, however, related, e.g., one class may subsume another. Supervised learning algorithms, on the other hand, assume that the classes are unrelated and cannot take any relationships into account. There is consequently a need for learning algorithms that predict classes in an ontology.

In this and the following chapters we will present learning algorithms for predicting functional annotations in an ontology and methods for evaluating the predictions made by these algorithms. We begin with an introduction to ontologies in general and the Gene Ontology (GO) in particular. This chapter also includes a discussion of the semantics of the Gene Ontology. We identify an issue that may complicate learning

[1]MIPS is an abbreviation for "Munich Information center for Protein Sequences".

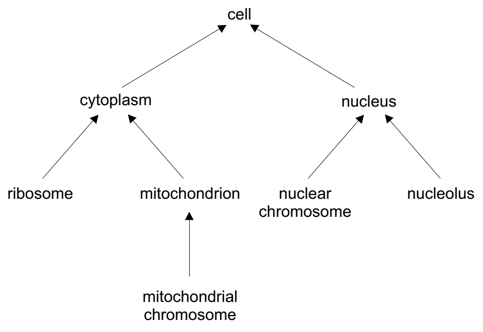


Figure 6.1: *A partial ontology for the part in a cell.*

in the GO and present a transformation algorithm that simplifies the ontology so that this problem does not occur.

6.2 What is an ontology?

The term ontology originates from philosophy where it is a branch of metaphysics concerned with the study of being and existence [189]. However, ontology may also denote a particular theory (developed within this field) that describes what entities exist [168], in which case we talk about *an* ontology. In artificial intelligence, the word is used more narrowly. Gruber [63] defines an ontology as "an explicit specification of a conceptualization". Put more simply, an ontology is a description of the concepts in a domain of interest and of the relationships between these concepts.

A simple example is shown in Figure 6.1. This ontology describes some of the parts in a cell, but is by no means complete. The edges (or arrows) describe **part-of**-relationships. The ontology states the following: The cytoplasm and the nucleus are both parts of a cell. The cytoplasm contains ribosomes and mitochondrions (amongst others). The nucleus contains nucleolus and (nuclear) chromosomes. Chromosomes are, however, not only found in the nucleus. The mitochondrions have also (mitochondrial) chromosomes.

Ontologies are important as they facilitate communication. They provide vocabularies with clearly defined semantics that can be used to share information between people as well as between computer systems. One application is found in databases. A gene database may for example store information about the subcellular location where a gene product is active. This information may, of course, be stored as a textual description. However, it will be practically impossible to select a set of genes according to the location in this case. A database system has only facilities for syntactic matching of strings and cannot interpret descriptions in natural language. The selection results would thus be very inaccurate if we tried to select genes on the basis of such descriptions. Hence, a database is most often built such that it refers only to terms in a controlled vocabulary. Such a vocabulary consists of a set of syntactically different terms or concepts that may be used to describe the domain of interest, e.g., the parts in the cell where a gene product may be active. The selection task reduces then to matching of terms, which is an easy task for a database system.

A controlled vocabulary is a partial ontology. It provides a syntax (a set of concepts), but no semantics (relationships). A complete ontology, on the other hand, provides a semantics by describing the relationships between the concepts. A semantics may not be needed in some situations, but often it will be required.

One example is integration of databases where an ontology may be used for combining similar information. We may for example retrieve information about genes and locations (where the gene products are active) from a variety of gene databases and try to combine this information. However, the databases must refer to the same location concepts if a computer is to perform this task automatically. In this case, it is not enough that the concepts are syntactically identical, they must also be semantically equivalent. Otherwise, if a concept has a different meaning in each database, it cannot be interpreted without knowledge of the context, i.e., the database from which this piece of information originated.

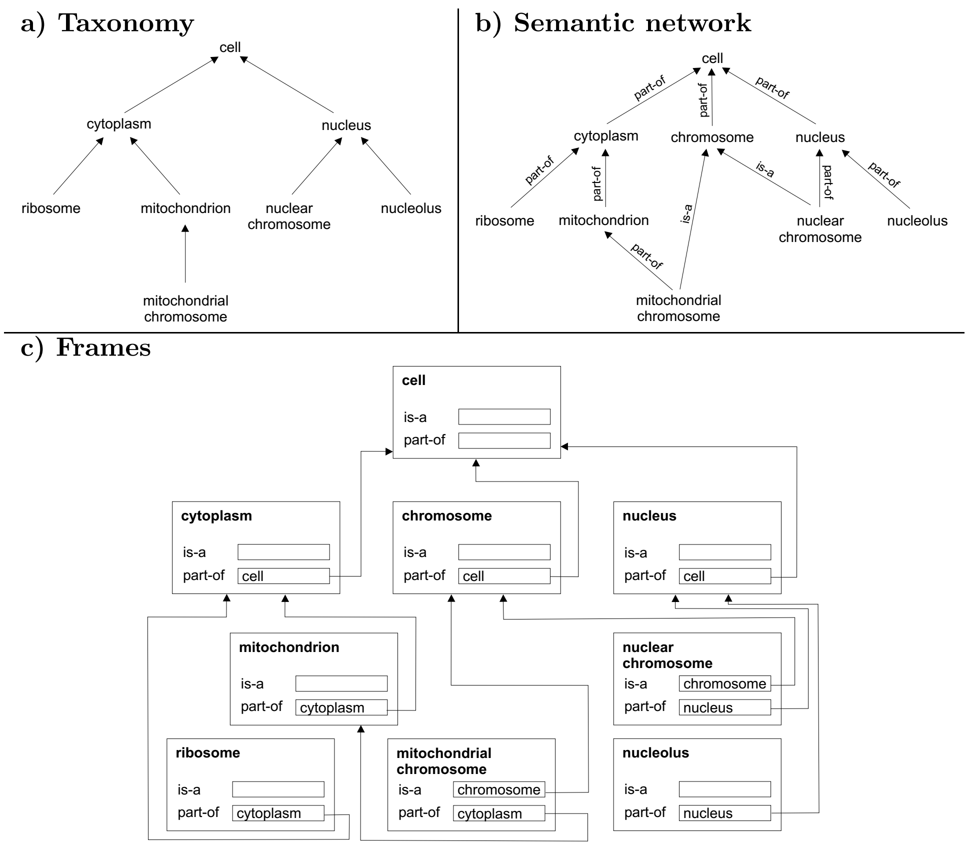
One may avoid such integration problems by using an ontology as a standard and let the databases refer to concepts in this ontology. In this case, it will be easy to combine information from different databases as they refer to the same concepts. However, it may not be possible to apply an ontology in this way. The databases that one would like to integrate will often be developed by different people and will most likely use different ontologies. The databases may therefore refer to semantically different concepts. Ontologies may still be very useful. One may, for example, try to define a thesaurus by comparing the structure of the ontologies. This thesaurus can then be used to translate the concepts in the different databases. Hence, ontologies facilitate integration of databases.

The information defined in an ontology is also useful for classification of genes. As mentioned above, a supervised learning system needs a set of classes that can be predicted. These classes are, of course, a controlled vocabulary. If these classes are related in some way, the learning algorithm has to take the relationships into account. A full ontology may therefore be required.

6.3 Representation of ontologies

An ontology may be represented in many different ways. Such representation forms are investigated in the subfield of artificial intelligence known as knowledge representation, which is still an active field of research. Here, we will only mention the most important forms.

Taxonomy. The simplest form is a taxonomy (e.g., [147]). This is a hierarchy which is organized according to the generality of the concepts. The most general concepts appear at the top and the most specific concepts at the bottom. If a concept



d) Description Logic

$$
\begin{aligned}
\text{Cell} &\equiv \cdots \\
\text{Cytoplasm} &\equiv \exists \text{partOf.Cell} \sqcap \cdots \\
\text{Chromosome} &\equiv \exists \text{partOf.Cell} \sqcap \cdots \\
\text{Ribosome} &\equiv \exists \text{partOf.Cytoplasm} \sqcap \cdots \\
\text{Mitochondrion} &\equiv \exists \text{partOf.Cytoplasm} \sqcap \cdots \\
\text{MitochondrialChromosome} &\equiv \exists \text{partOf.Mitochondrion} \sqcap \text{Chromosome} \sqcap \cdots \\
\text{Nucleus} &\equiv \exists \text{partOf.Cell} \sqcap \cdots \\
\text{NuclearChromosome} &\equiv \exists \text{partOf.Nucleus} \sqcap \text{Chromosome} \sqcap \cdots \\
\text{Nucleolus} &\equiv \exists \text{partOf.Nucleus} \sqcap \cdots
\end{aligned}
$$

Figure 6.2: *Ontology representations*

A occurs below another concept B, we say that A is subconcept or a child of B, and B is a superconcept or parent of A. A concept subsumes all of its subconcepts. Hence, an object (e.g., a gene) that belongs to one of the subconcepts must also belong to the concept as well. The concepts in a taxonomy have usually only one immediate superconcept so that the taxonomy forms a tree. A typical example is a set of classes organized in an inheritance hierarchy according to a **is-a**-relation. Figure 6.1 gives another example, which is repeated in Figure 6.2a. This ontology is organized according to a **part-of**-relation so that a subconcept is a part of a superconcept.

Semantic networks. A taxonomy is a very restricted form of representation, and many properties and relationships cannot be expressed in such a language. For example, there are two different kinds of chromosomes in the taxonomy in Figure 6.2a. Both of these are subconcepts of the more general concept chromosome, but there is no way to represent this information since only one kind of relationship can be expressed. Moreover, a node may have only one direct parent.

Semantic networks (e.g., [102]) can be considered as generalizations of taxonomies where more than one kind of relationship can be represented. Such a network is a graph where the nodes represent concepts, and the edges represent relations. The edges may form cycles, and a node may have more than one parent. One example is shown in Figure 6.2b.

Frames. One alternative to semantic networks is the so-called frames (see e.g., [191]). A frame represents a concept and is similar to a class in object-oriented methodology. A frame has a set of slots, and each slot may hold one or more value(s). A slot may describe either a property of the frame or a relationship to another frame. In the last case, the slot value is a pointer to the other frame. A set of frames is displayed in Figure 6.2c. Each of these frames has an **is-a**-slot and a **part-of**-slot, which are pointers to some other frames.

Description Logics. A serious drawback with frames and semantic networks is their lack of formal semantics. An ontology represented in these languages may therefore be ambiguous. One example is a semantic network that contains the concepts A, B, and C where concept C may be an instance of concept B, and concept B may be a part of A. In this case, it is not clear how C and A are related or if they are related at all. One would assume that C may be a part of A, but this cannot be deduced from the network alone. A rule is needed in order to define how the relationships should be interpreted. Moreover, one may want to check for inconsistencies, determine whether two concepts are equivalent, or detect whether one concept subsumes another. However, automated reasoning is not feasible without a clear semantics.

These problems may be avoided by either defining a semantics for the representation language in first-order logic (cf. [35]), or by representing the ontology directly in first-order logic. However, reasoning in first-order logic, such as determining whether

Description logic	First-order logic	
C	$C'(x)$	(Concept)
R	$R'(x,y)$	(Relation)
$C \sqcap D$	$C'(x) \wedge D'(x)$	(Intersection)
$\forall R.C$	$\forall y.(R'(x,y) \rightarrow C'(y))$	(Value restriction)
$\exists R.C$	$\exists y.(R'(x,y) \wedge C'(y))$	(Existential quantification)

Table 6.1: *Translation from description logic to first-order logic.*

concept A is a subconcept of concept B, is undecidable[2] (see e.g.,[170]) since this involves testing if the definition of B is a logic consequence of the definition of A. First-order logic is consequently not a better alternative if a computer is to reason over the ontology.

The full representational power of first-order logic is often not necessary for representing an ontology, and only a small part may be used instead. A *description logic* is a fragment of first-order logic that has sufficient expressive power to represent an ontology and renders reasoning computationally feasible. Many description logics permit reasoning in polynomial time, but research is also pursued on more expressive description logics where theoretical results show that inference is computationally intractable [4]. In this case, optimization techniques are used in order to make the task more tractable.

There are many different description logics, but they are all quite similar and differ mainly in the constructors that they provide. In Figure 6.2d, the $\mathcal{AL}$-language introduced by Schmidts-Schlauß and Smolka [152] is used. A description in this language is built from elementary descriptions and constructors. An elementary description can be either a *concept* or a *role*. A role is similar to a relation in our terminology. The elementary descriptions are represented by symbols such as C and D for concepts and R for relations. There are no variables as in first-order logic, but variables exist implicitly. A concept C corresponds to a unary predicate $C'(x)$ and a role R to a binary predicate $R'(x,y)$. The language includes constructors such as *intersection* ($C \sqcap D$) and *value restriction* ($\forall R.C$). There are also extensions of the language that include *unions* ($C \sqcup D$) and *existential quantification* ($\exists R.C$). An expression in this language can be translated into first-order logic (see Table 6.1). For example, $\forall R.C$ may be translated into $\forall y.(R'(x,y) \rightarrow C'(y))$ in first-order logic, and $\exists R.C$ may be translated into $\exists y.(R'(x,y) \wedge C'(y))$.

A partial description of the example ontology is given in Figure 6.2d. In order to make this ontology complete, other distinguishing properties would have to be

[2]Logic consequence in first-order logic is, strictly speaking, semi-decidable. We may develop an algorithm that terminates if ϕ follows from a set of formulae $\phi_1, \ldots, \phi_n$ (i.e., if $\phi_1, \ldots, \phi_n \models \phi$ holds). However, it is not possible to develop an algorithm that also terminates when ϕ does not follow from a set of formulae $\phi_1, \ldots, \phi_n$ (if $\phi_1, \ldots, \phi_n \not\models \phi$ is true). We consider the problem – whether concept A is a subconcept of concept B or not – to be undecidable since we need an algorithm that terminates in both cases in order to answer this question.

described. Notice that **is-a**-relations can be inferred directly from the descriptions and are not represented directly in this language[3]. For example, Chromosome follows from (subsumes) NuclearChromosome.

6.4 Ontologies for characterization of gene function

Ontologies have a long history in biology and medicine. Most famous is Carl Linnaeus' taxonomy from the mid-18th century. This taxonomy introduced a novel way of grouping organisms – an activity, which was performed more or less arbitrarily at that time. In Linnaeus' taxonomy, plants and animals were divided into groups according to biological traits. The ontology has since his time undergone several modifications, but the principle of grouping organisms on the basis of shared traits still remains. The current version consists of 7 levels: kingdom (plants/animals), phylum, class, order, family, genus, and species. With this ontology, Linnaeus also introduced the naming convention that is used in biology. Organisms are referred to by their genus and species name. For example, *Saccharomyces cerevisiae* (yeast) belongs to the genus *saccharomyces* and the species *cerevisiae*.

Other examples of ontologies in biomedicine are the Medical Subject Heading (MeSH) system [28], which is a controlled vocabulary for searching the biomedical literature and the Unified Medical Language System (UMLS) [94], which attempts to improve retrieval and integration of biomedical information by defining a thesaurus for various biomedical vocabularies.

Ontologies that characterize the function of gene products are a more recent addition to biology. These are typically developed for bioinformatics databases in order to provide a vocabulary for querying a database according to function. One of the first attempts to build an ontology was by Riley [143], who cataloged the genes in *Escherichi coli* and devised an ontology in this process. Some other attempts are:

- EcoCyc [81] (A frame approach)
- MIPS [57] (A taxonomy)
- The Gene Ontology (GO) [29] (Several semantic networks)
- KEGG [125] (A taxonomy)
- WIT [128] (A taxonomy)

These ontologies vary with regard to representation language and detail level, i.e., how specific concepts are modeled. They may also be species specific or species independent. Moreover, the notion of function is different in some of these ontologies. The term "function" has a rather vague meaning when applied to gene products, and it encompasses several different aspects of the activity of a gene product. For example, a protein may take part in a particular reaction on a molecular level, but

[3]However, such relationships are represented explicitly in some description logics [51, 171].

the protein may also play a particular part in a biological objective on a cellular level. Some of these ontologies make a clear distinction between these aspects and create a subontology for each aspect (e.g., the Gene Ontology). Others consider mainly one aspect or ignore this distinction.

The most comprehensive of these ontologies is the Gene Ontology. It also uses a more complex representation language than most of the others, which are simple hierarchical trees. We will thus focus on this ontology and discuss it in more detail in the next sections. The learning methods that will be introduced are nonetheless suitable for taxonomies such that most of these ontologies can be used directly with these methods. Rison et al. [144] provide a comparison of the mentioned ontologies and several others.

6.5 The Gene Ontology

The Gene Ontology (GO) Consortium was established to develop a shared and structured vocabulary for describing the function of the gene products in any organism [29]. The ontology developed by the consortium is not really a single ontology, but consists of three subontologies where each subontology describes a particular aspect of a gene product. These aspects are as follows:

- **Biological process**: refers to the objective to which a gene or a gene product contributes.
- **Molecular function**: refers to the biochemical activity, i.e., what the gene product does on the biochemical level, and
- **Cellular component**: refers to the subcellular location where a gene product is active.

Each subontology is represented as semantic network where only two kinds of relationships are expressed. Two concepts may be related through a **is-a-** or a **part-of**-relationship. A concept may have more than one parent and may be related to these parents through different kinds of relationships, but no cycles are allowed in the network. Hence, each subontology forms a Directed Acyclic Graph (DAG).

A part of the biological process ontology is shown in Figure 6.3. We will consider this subontology (i.e., biological process) exclusively in the experiments in Chapter 10 since only this aspect is considered relevant for predicting gene function from temporal expression patterns.

The GO Consortium provides not only ontologies, but also annotations for several model organisms [30]. An annotation is a description of a role that a gene plays in a cell. It may just be an statement written in a natural language that explains the function of the gene, or it may be an association between a gene and a concept taken from an ontology. The annotations created by the GO consortium are associations between a gene and a concept in one of the subontologies. A gene may have an annotation for each of the subontologies. However, it may play several different roles

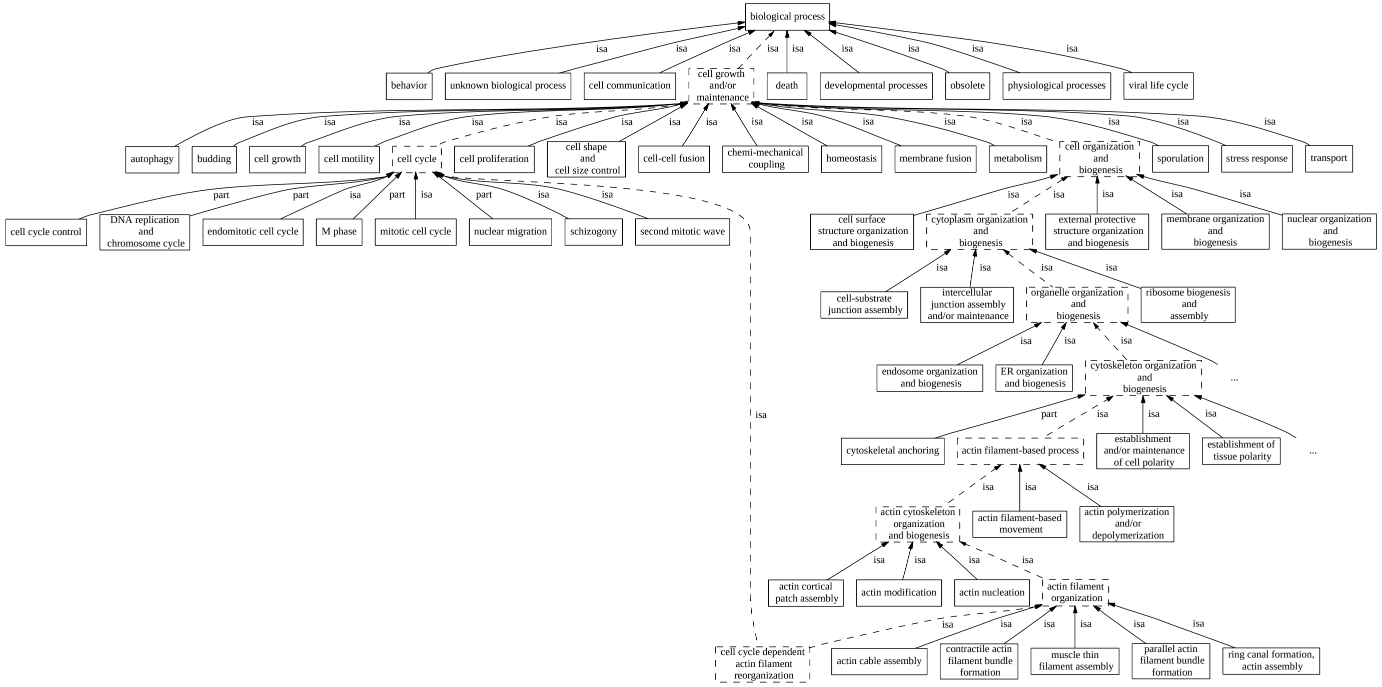

biological process
behavior
unknown biological process
cell communication
cell growth and/or maintenance
death
developmental processes
obsolete
physiological processes
viral life cycle
autophagy
budding
cell growth
cell motility
cell cycle
cell proliferation
cell shape and cell size control
cell-cell fusion
chemi-mechanical coupling
homeostasis
membrane fusion
metabolism
cell organization and biogenesis
sporulation
stress response
transport
cell cycle control
DNA replication and chromosome cycle
endomitotic cell cycle
M phase
mitotic cell cycle
nuclear migration
schizogony
second mitotic wave
cell surface structure organization and biogenesis
cytoplasm organization and biogenesis
external protective structure organization and biogenesis
membrane organization and biogenesis
nuclear organization and biogenesis
cell-substrate junction assembly
intercellular junction assembly and/or maintenance
organelle organization and biogenesis
ribosome biogenesis and assembly
endosome organization and biogenesis
ER organization and biogenesis
cytoskeleton organization and biogenesis
...
cytoskeletal anchoring
actin filament-based process
establishment and/or maintenance of cell polarity
establishment of tissue polarity
...
actin cytoskeleton organization and biogenesis
actin filament-based movement
actin polymerization and/or depolymerization
actin cortical patch assembly
actin modification
actin nucleation
actin filament organization
cell cycle dependent actin filament reorganization
actin cable assembly
contractile actin filament bundle formation
muscle thin filament assembly
parallel actin filament bundle formation
ring canal formation, actin assembly
isa
part

Figure 6.3: *A part of the process ontology (Rev. 2.577 - 24-Sep-2002).*

in the cell. A gene may therefore have not only one, but several annotations for each subontology.

Originally, annotations were available only for *Drosophila melanogaster* (fruitfly), *Saccharomyces cerevisiae* (yeast), and *Mus musculus* (mouse) from the GO Consortium. Annotations for more organisms (such as *Arabidopsis thaliana*) have recently become available as other model organism groups have joined the consortium. However, none of these organisms have a completely annotated genome. A lot of the genes are still uncharacterized, and most of the available knowledge about characterized genes has not been formalized as annotations. Automatic prediction of the gene function is therefore of great importance.

6.6 Interpretation of the Gene Ontology

The subontologies do not have a formal semantics as they are semantic networks. However, the GO consortium has defined several rules for interpreting the networks.

6.6.1 The True Path Rule

Most important is the *True Path Rule* that says that "The pathway from a child term to its top-level parent(s) must always be true" [30, 42]. A gene annotated to a concept is therefore also annotated to any parent concept. This holds for all paths such that if a gene is annotated to *cell cycle dependent actin filament reorganization* in Figure 6.3, it belongs to both *cell cycle* and *actin filament organization* as well as their parents (as marked with dashed lines in Figure 6.3).

A formalization of the True Path Rule is given at top of Figure 6.4. Notice that there is no difference between the **is-a**- and **part-of**-relationships with regard to this rule. Both relationships define one concept to be *less general* than another concept in some way, and the True Path Rule says that the genes that are annotated to one concept also belong to the concepts that are more general. Hence, it seems that we may ignore the difference between the two relations and consider the concepts to be connected by *less general*-edges.

6.6.2 Rules on the concepts

Besides the True Path Rule, the GO consortium has defined several logical rules, which are shown in Figure 6.4. These rules are important as they provide a semantics for the **is-a**- and **part-of**-relationships between the concepts. Unfortunately, there are some details that are not defined by theses rules, and they seem to complicate the interpretation a bit. In particular, it is not clear how these rules should be interpreted with regard to the True Path Rule

The two first make the **is-a**- and **part-of**-relations transitive and represent no need for concern. Rule 4 is more difficult, but not really a problem. It states, for example, that a *synaptonemal complex* is not necessarily a part of a *chromosome*. The True

True Path Rule:

if **anno**(g, c) and **is-a**(c, d) then **anno**(g, d), for all $g \in G$ and $c, d \in V$ $(*)$

if **anno**(g, c) and **part-of**(c, d) then **anno**(g, d), for all $g \in G$ and $c, d \in V$ $(**)$

Relationships between the terms:

1. Transitivity of **is-a**

   if **is-a**(c, d) and **is-a**(d, e) then **is-a**(d, e), for all $c, d, e \in V$

   Example:

   Terminal O-glycosylation is an instance of *terminal glycosylation.*
   Terminal glycosylation is an instance of *protein glycosylation.*
   Terminal O-glycosylation is an instance of *protein glycosylation.*

2. Transitivity of **part-of**

   if **part-of**(c, d) and **part-of**(d, e) then **part-of**(d, e), for all $c, d, e \in V$

   Example:

   Laminin-1 is a part of *basement lamina.*
   Basement lamina is a part of *basement membrane.*
   Laminin-1 is a part of *basement membrane.*

3. An instance inherits all parts from their parents

   if **part-of**(c, d) and **is-a**(e, d) then **part-of**(c, e), for all $c, d, e \in V$

   Example:

   Protein-nucleus import, docking is a part of *protein-nucleus import.*
   Ribosomal protein-nucleus import is an instance of *protein-nucleus import.*
   Protein-nucleus import, docking is a part of *ribosomal protein-nucleus import.*

4. A part of an instance of a concept need not be a part of this concept

   if **part-of**(c, d) and **is-a**(d, e) then **part-of**(c, e), for all $c, d, e \in V$, does not necessarily hold.

   Example:

   Synaptonemal complex is a part of *meiotic chromosome.*
   Meiotic chromosome is an instance of *chromosome.*
   Synaptonemal complex is **not necessarily** a part of *chromosome.*

V is a set of terms/nodes, and G is a set of genes. **part-of**(x, y) means that $x \in V$ is a part of $y \in V$. **is-a**(x, y) means that $x \in V$ is an instance of $y \in V$. **anno**(g, x) means that gene $g \in G$ is annotated to $x \in V$.

Figure 6.4: *The semantics of the Gene Ontology (from The GO Usage Guide [31]).*

Path Rule, however, says that the genes annotated to *synaptonemal complex* belong to *meitotic chromosome* by (**) and thus also to *chromosome* by (*). This may seem odd, but the rules are consistent and seem to describe the biology accurately. Even though a *synaptonemal complex* is not a part of every chromosome, the genes that are active in *synaptonemal complex* are also active in some chromosomes and should be annotated to the chromosome concept as well. Hence, one should really distinguish between the ordering of the genes and the ordering of cellular components (or molecular functions or biological processes). Rule 4 applies to ordering of cellular components, while True Path Rule applies to ordering of annotations. Thus Rule 4 may be ignored when we consider the ontology with respect to gene annotations.

The main concern is Rule 3, which states that a concept inherits the parts of its parents. This principle is not completely novel – it is well-established in object-oriented methodology. However, when it is applied to an ontology, it complicates the ordering of the concepts and thus also the ordering of the genes.

6.6.3 The inheritance problem

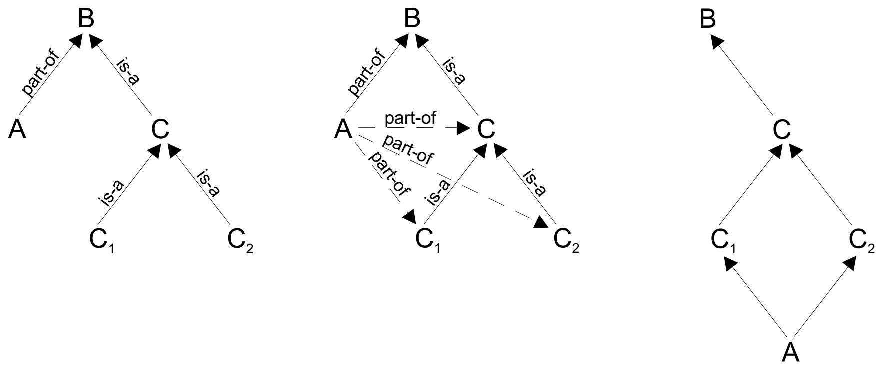
A natural interpretation of a DAG is that the order of concepts is defined directly by edges. There is only a relationship between two concepts if there is a path between them. All concepts that are more specific than a given concept should therefore appear as its children. We call this principle the Direct Correspondence Assumption (DCA).

Rule 3, however, violates this assumption. For example, let A be a part of B and C an instance of B as shown in Figure 6.5. All concepts more specific than C should then appear below C by the DCA. However, A is a part of C by Rule 3 and does not appear below C.

The violation of the DCA has consequences for the ordering of the genes since the True Path Rule as formalized in Figure 6.4 is defined on the **part-of**-relation derived from Rule 3. Hence, the subset inclusion order on the genes defined by the True Path Rule does not correspond to the DAG either. The set of genes annotated to A is for example a subset of the genes of C.

This will create a serious problem for a learning algorithm since A and C are siblings in the DAG, and we would normally try to distinguish between them. The sets of genes in A and in C should therefore be disjoint or almost disjoint (Recall that a gene may have more than one annotation such that it may be annotated to both A and C). However, since C contains all of the genes that are annotated to A, it is not possible to distinguish between them.

The problem may be handled in different ways. Either one may build the inheritance semantics into the learning algorithm, or one may transform the ontology such that it respects the DCA. The first option does not seem very attractive. It would make the learning algorithm very dependent on the GO, and we would prefer that the algorithm could be used on other ontologies and in other domains. Moreover, this semantics would complicate the implementation and analysis of the learning algorithm. Transformation is therefore a better option. It simplifies the problem, while nothing is lost. Consequently, we introduce an algorithm for making such a transformation in



(a) Original DAG (b) DAG with Rule 3 (c) Transformed DAG

Figure 6.5: *Transformation of inheritance in the Gene Ontology.* The figure demonstrates the transformation of the ontology in (a). The part-relationships induced by the inheritance rule are shown as dashed arrows in (b). The result of the transformation is the ontology in (c). This maintains the same set-inclusion order on the annotations as (b). Note that the transformed ontology does not distinguish between **is-a-** and **part-of**-relationships. Thus no labels are associated with the arrows in (c).

the next section.

6.7 Transformation of the Gene Ontology

Before we introduce the transformation algorithm, we would like to give a warning. It seems that the current version of the Gene Ontology does not implement the inheritance rule or at least it defines relationships that are in conflict with this rule. This will be discussed more closely in Section 6.8. However, the algorithm may give an unexpected result if it is applied on the current ontology. Thus, it should not be used with this version.

The transformation problem is illustrated in Figure 6.5 where A is a **part-of** B. According to Rule 3, A must also be a **part-of** C, C_1 and C_2. So the **part-of**-relation derived from the DAG violates the DCA. Our solution is to move A below all leaf nodes (i.e., C_1 and C_2) in the **is-a**-subhierarchy below C. Then, there is a path from A to each of these nodes. The relationships between A and C_1 and C_2 follow directly for the DAG, and the relationships with C and B follow by transitivity.

In order to present the algorithm and prove correctness and completeness, we need to introduce several definitions. We begin by formalizing what is meant by a GO-DAG.

Definition 6.1 (GO-DAG). Let V be a set of nodes, $\top \in V$ a root node, $I \subseteq V \times V$

a set of **is-a**-edges, and $P \subseteq V \times V$ a set of **part-of**-edges. A GO-DAG is a tuple $\langle V, \top, I, P\rangle$ if I and P are disjoint sets, and $\langle V, \top, P \cup U\rangle$ forms a rooted DAG (see Definition A.16). ■

As the algorithm manipulates the actual edges in the DAG, we distinguish between the edges in the DAG and the relations that are derived from the edges. The following definitions correspond to the rules in Figure 6.4.

Definition 6.2 (is-a). Let $D = \langle V, \top, I, P\rangle$ be a GO-DAG. Then, for all $c, d \in V$, **is-a**(c, d) iff[4]

1. $\langle c, d\rangle \in I$, or
2. **is-a**(c, e) and **is-a**(e, d), for some $e \in V$ ■

Definition 6.3 (part-of). Let $D = \langle V, \top, I, P\rangle$ be a GO-DAG. Then, for all $c, d \in V$, **part-of**(c, d) iff

1. $\langle c, d\rangle \in P$, or
2. **part-of**(c, e) and **is-a**(d, e), for some $e \in V$, or
3. **part-of**(c, e) and **part-of**(e, d), for some $e \in V$ ■

These definitions imply that the **is-a**-relation and the **part-of**-relation are transitive. However, they do not define a transitive closure over all edges since Rule 4 implies that this may not hold. Hence, there may not be any relationship between two concepts if they are connected by a path that consists of both **is-a**- and **part-of**-edges.

The True Path Rule, however, ignores the difference between the **is-a**- and the **part-of**-relations, and we may complete the transitive closure for the genes by iterative applications of $(*)$ and $(**)$. In order to make this more evident and to simplify the proofs, we introduce a *strictly less general* relation with a full transitive closure on top of the **is-a**- and the **part-of**-relations. The True Path Rule may then be defined on this relation.

Definition 6.4 (slg). Let $D = \langle V, \top, I, P\rangle$ be a GO-DAG. Then, for all $c, d \in V$, **slg**(c, d) iff

1. **is-a**(c, d), or
2. **part-of**(c, d), or
3. **slg**(c, e) and **slg**(e, d), for some $e \in V$ ■

The next lemma proves that the same subset inclusion order on the genes is obtained if the True Path Rule is defined on this relation, instead of the **is-a**- and the **part-of**-relations.

[4]Note that "if and only if" is abbreviated with "iff", which is the custom in the computer science literature.

Lemma 6.1. *Let* $\textbf{slg}$ *be a relation on the DAG as in Definition 6.4. Then the following rule is equivalent to the True Path Rule (as defined in Figure 6.4):*

if $\textbf{anno}(g, c)$ *and* $\textbf{slg}(c, d)$ *then* $\textbf{anno}(g, d)$*, for all* $g \in G$ *and* $c, d \in V$ ■

Proof.

⇐: Assume that the rule does not follow from the True Path Rule. Then there must be a situation where the rule does not hold while True Path Rule does. In this situation there must be a gene $g' \in G$ and some concepts $c', d' \in V$ that do not satisfy the rule such that $\textbf{anno}(g', c')$ and $\textbf{slg}(c', d')$ are true, but $\textbf{anno}(g', d')$ is not. We now prove by induction on **slg** that $\textbf{anno}(g', d')$ must follow from the True Path Rule and $\textbf{anno}(g', c')$. So this is impossible.

Basis: If $\textbf{slg}(c', d')$ follows from the two first options in Definition 6.4, we have either $\textbf{is-a}(c', d')$ or $\textbf{part-of}(c', d')$. Then $\textbf{anno}(g', d')$ follows directly from $(*)$ or $(**)$.

Step: If $\textbf{slg}(c', d')$ follows from the last option in Definition 6.4, we may select (without loss of generality) an e' such that $\textbf{slg}(e', d')$ corresponds to $\textbf{is-a}(e', d')$ or $\textbf{part-of}(e', d')$. $\textbf{anno}(g', e')$ must follow from $\textbf{slg}(c', e')$ and $\textbf{anno}(g', c')$ by the induction hypothesis. Then $\textbf{anno}(g', d')$ follows from the True Path Rule and $\textbf{anno}(g', e')$.

⇒: Assume that $(*)$ or $(**)$ do not follow from the rule. As before, there must be a $g' \in G$ and some $c', d' \in V$ where $\textbf{anno}(g', c')$ is true and $\textbf{anno}(g', d')$ is false. Moreover, either $\textbf{is-a}(c', d')$ or $\textbf{part-of}(c', d')$ must hold. Hence, $\textbf{slg}(c', d')$ must be true. This means that $\textbf{anno}(g', d')$ follows from the rule, and we have obtained a contradiction. □

The previous lemma states that we need only to consider the **slg**-order as far as the annotations are concerned. Thus, it is sufficient to transform the GO-DAG into a DAG $D' = \langle V', \top', E' \rangle$ with DCA-compliant *strictly less general* relation $\textbf{slg}'$ that is equivalent to **slg**. The following definition defines this relation, which obviously obeys the DCA since it adds only transitivity.

Definition 6.5 ($\textbf{slg}'$). Let $D = \langle V, \top, E \rangle$ be a DAG. Then, for all $c, d \in V$, $\textbf{slg}(c, d)$ iff

1. $\langle c, d \rangle \in E$, or
2. $\textbf{slg}'(c, e)$ and $\textbf{slg}'(e, d)$, for some $e \in V$ ■

We are now ready to introduce the transformation algorithm, which is shown in Algorithm 6.1. The algorithm transforms a GO-DAG D' = $\langle V, \top, I, P \rangle$ into a rooted DAG $D' = \langle V', \top', E' \rangle$ by traversing DAG recursively starting at the root $\top$. It

TransformOntology:
Input: A GO-DAG D = $\langle V, \top, I, P\rangle$, a node d, and a list L of **part-of**-nodes to be moved.
Output: A rooted DAG $D' = \langle V', \top', E'\rangle$.

```
$C_I(d) = \{c \mid \langle c,d\rangle \in I\}$
$C_P(d) = \{c \mid \langle c,d\rangle \in P\}$
$V' = \emptyset$ and $E' = \emptyset$ and $L' = \emptyset$
if $d$ has not been visited then
   mark $d$ as visited
   $V' = \{d\}$ and $L' = C_P(d)$
   for all $c \in C_P(d)$ do
      $\langle V'', \top', E''\rangle = $ Transform$(D, c, \emptyset)$
      $V' = V' \cup V''$ and $E' = E' \cup E''$
   end for
   $E' = E' \cup \{\langle c,d\rangle \mid c \in C_I(d)\}$        {Insertion of is-a-children}
end if
if $C_I(d) = \emptyset$ then {$d$ is a leaf node or has only part-of-children}
   $E' = E' \cup \{\langle c,d\rangle \mid c \in L \cup L'\}$        {Insertion of part-of-children}
else
   for all $c \in C_I(d)$ do
      $\langle V'', \top', E''\rangle = $ Transform$(D, c, L \cup L')$
      $V' = V' \cup V''$ and $E' = E' \cup E''$
   end for
end if
return$(\langle V', d, E'\rangle)$
```

Algorithm 6.1: *An algorithm for transforming the GO-DAG to a DAG that corresponds to a subset inclusion order on the genes.*

follows both **is-a**- and **part-of**-edges. However, the edges are treated differently. **is-a**-edges are inserted immediately into the new DAG when a node is visited, while **part-of**-children are passed downwards in the **is-a**-subhierarchy until a leaf node or a node without **is-a**-children is reached. The **part-of**-children are then attached below this node.

Note that a node may have several **is-a**-parents, and the **part-of**-nodes from these parents have to be attached at the bottom of **is-a**-subhierarchy of the node. It is therefore necessary to travel down the **is-a**-subhierarchy once for each parent. However, certain tasks should not be repeated in this subhierarchy. Neither the insertion of **is-a**-edges nor the visiting and the passing of **part-of**-children should be performed more than once. These tasks are consequently performed only the first time a node is visited.

The algorithm maintains certain invariants, which are needed in the correctness and the completeness proofs. First, a node c is only in the **part-of**-node list L of node x if c is a **part-of** x. Second, $\mathbf{slg}'(c, x)$ will be derivable from the new DAG in this case.

Lemma 6.2. *Let $D = \langle V, \top, I, P\rangle$ be a GO-DAG. If Algorithm 6.1 is initially called as **Transform**$(D, \top, \emptyset)$ then for any $x \in V$ called recursively as **Transform**(D, x, L), it holds that $c \in L$ implies **part-of**(c, x).* ■

Proof. Each $c \in L$ must have been added at some node y on the path from the $\top$ to x since L is initially empty, and nodes are only added to L in the recursive call in line 17. Nodes are added to L through L'. This set is equal to $C_P(y)$ when y is visited for the first time and empty otherwise. This means that c must be in $C_P(y)$ such that **part-of**(c, y) holds.

Moreover, the path between y and x must consist of only **is-a**-edges since if there were two nodes u and v on this path, and these were connected by an edge $\langle u, v\rangle \in P$, u would be in $C_P(v)$. Thus, u would be called in line 8, and c would not be in L. Hence, **is-a**(x, y). Then **part-of**(c, x) follows from option 2 in Definition 6.3. □

Lemma 6.3. *Given a GO-DAG $D = \langle V, \top, I, P\rangle$, a node $x \in V$, and a subset $L \subseteq V$ of nodes, a call **Transform**(D, x, L) to Algorithm 6.1 creates a DAG such that $\mathbf{slg}'(y, x)$ holds for all $y \in L$. Moreover, for all $z \in V$ where **is-a**(z, x), $\mathbf{slg}'(y, z)$ holds for all $y \in L$ as well.* ■

Proof. By induction on the **is-a**-edges.

Basis: If $C_I(x) = \emptyset$ so that x is a leaf node or has no **is-a**-children, the algorithm inserts $\langle y, x\rangle$ into E' for each $y \in L$ (line 14). Hence, $\mathbf{slg}'(y, x)$ (for each $y \in L$) follows from Definition 6.5.

Step: If $C_I(x) \neq \emptyset$, $\mathbf{slg}'(y, z)$ holds for all $y \in L$ and $z \in C_I(x)$ by the induction hypothesis and the fact that **Transform**(D, z, L) is called in line 17. Moreover, the first time x is visited $\langle z, x\rangle$ is inserted into E' for all $z \in C_I(x)$. So $\mathbf{slg}'(z, x)$ holds for each $z \in C_I(x)$. By the transitivity of $\mathbf{slg}'$, $\mathbf{slg}'(y, x)$ holds for all $y \in L$.

Since any $z \in V$ where **is-a**(z, x), is called recursively from x with a superset of L (line 17), $\mathbf{slg}'(y, z)$ must also hold for any $y \in L$ and any of these z's. □

The two of the options in Definition 6.3 are recursive. This would complicate the completeness proof since both options would have to be considered at once. However, the completeness proof may be simplified by noting that the definition is really stratified. This means that the tuples in the **part-of**-relation can be found by applying the options in sequence. First a set is found from option 1. Then a larger set is found by applying option 2 of this set, and so on.

Definition 6.6 (Stratified definition). The definition of a relation R is stratified if R can be divided into a collection of sets $R_1, \dots, R_k$ ($k \geq 1$) such that $R = R_0 \cup \cdots \cup R_k$, and R_i ($1 \leq i \leq k$) is derived from $R_1 \cup \cdots \cup R_i$ with option i. ■

Notice that a tuple in R_i can be derived for another tuple in R_i according this definition. Hence, it allows recursive applications of option i.

The main idea behind the next proof is as follows: If **part-of**(c', d') can be derived from option 2 in Definition 6.3 such that the following rule holds,

$$\text{if } \textbf{part-of}(c', e') \text{ and } \textbf{is-a}(d', e') \text{ then } \textbf{part-of}(c', d') \qquad \text{(r1)}$$

and **part-of**(c', e') can be derived from option 3 (in Definition 6.3) such that this rule is true,

$$\text{if } \textbf{part-of}(c', f') \text{ and } \textbf{part-of}(f', e') \text{ then } \textbf{part-of}(c', e') \qquad \text{(r2)}$$

then we may unfold the first rule with the second rule such that we obtain:

$$\text{if } \textbf{part-of}(c', f') \text{ and } \textbf{part-of}(f', e') \text{ and } \textbf{is-a}(d', e') \text{ then } \textbf{part-of}(c', d') \qquad \text{(r3)}$$

In this case, option 2 implies that this rule holds as well.

$$\text{if } \textbf{part-of}(f', e') \text{ and } \textbf{is-a}(d', e') \text{ then } \textbf{part-of}(f', d') \qquad \text{(r4)}$$

This means that we may fold Rule r3 such that we obtain:

$$\text{if } \textbf{part-of}(c', f') \text{ and } \textbf{part-of}(f', d') \text{ then } \textbf{part-of}(c', d') \qquad \text{(r5)}$$

that corresponds to option 3. Hence, there is an alternate derivation of **part-of**(c', d') where option 2 is applied before option 3.

Lemma 6.4. *Definition 6.3 is stratified.* ■

Proof. The **part-of**-relation can be decomposed as follows[5]:

$$\textbf{part-of} = Q_1 \cup Q_2 \cup Q_3 \cup Q_4 \cup Q_5 \cup \cdots$$

where Q_i is derived from $S_i = Q_1 \cup \cdots \cup Q_i$ and option $o(i)$. The function $o(i)$ is defined as

$$o(i) = \begin{cases} 1 & \text{if } i = 1 \\ (i \bmod 2) + 2 & \text{otherwise} \end{cases}$$

This means that Q_1 is derived from option 1; $Q_2, Q_4, Q_6, \dots$ are derived from option 2; and $Q_3, Q_5, Q_7, \dots$ are derived from option 3.

[5]This follows from fixpoint semantics of definite logic programs. For example, S_i is equal to $T_i^{(n)}(S_{i-1})$ where T_i is an immediate consequence operator for option $o(i)$, and $T_i^{(n)}$ denotes n applications of this operator. This operator is obviously a bit different from the immediate consequence operator, which is usually applied in logic programming since each option is handled separately. However, it will still reach a fixpoint. For more details see [123, 96].

From this decomposition, it follows that $S_i \subseteq$ **part-of**. We will show that $Q_j \subseteq S_3$ for $j \geq 4$ such that **part-of** $= S_3 = Q_1 \cup Q_2 \cup Q_3$. Definition 6.3 is therefore stratified.

Assume that $\langle c, d\rangle$ is Q_4, but not in Q_1, Q_2, or Q_3 (i.e., $\langle c, d\rangle \notin S_3$) such that $Q_4 \not\subseteq S_3$. This means that **part-of**(c, d) must be derived from option 2 such that **part-of**(c, e) and **is-a**(d, e) must hold for some $e \in V$. $\langle c, e\rangle$ cannot be in S_2 since $\langle c, d\rangle$ would be in S_2 in this case. So, $\langle c, e\rangle$ must be in either Q_3 or Q_4.

We may assume that $\langle c, e\rangle$ is in Q_3. The reason is that if $\langle c, e\rangle$ in Q_4, we may prove that there is alternate derivation of $\langle c, d\rangle$, which is based on a tuple in S_3. The proof is by induction and the hypothesis is as follows: If there is a derivation of **part-of**(c, d) such that $\langle c, d\rangle \in Q_4$ then there is an $x \in V$ such that $\langle c, x\rangle \in S_3$ and **is-a**(d, x) is true.

Basis: We know that **part-of**(c, d) follows from **part-of**(c, e) and **is-a**(d, e). So, if $\langle c, e\rangle \in S_3$, we may set $x = e$, and the hypothesis holds.

Step: If $\langle c, e\rangle \in Q_4$, **part-of**$(c, e)$ must be derived from option 2. By the induction hypothesis, there must be some $x \in V$ where $\langle c, x\rangle \in S_3$ and **is-a**(e, x). Since **is-a**(d, e) is true, we have by the transitivity of **is-a** that **is-a**(d, x). Hence, the hypothesis holds.

Thus, assume that $\langle c, e\rangle \in Q_3$. This means that **part-of**(c, e) must follow from option 3. Then **part-of**(c, f) and **part-of**(f, e) must hold for some $f \in V$. Both $\langle c, f\rangle$ and $\langle f, e\rangle$ must be in S_3 (since $\langle c, e\rangle \in S_3$). However, we may assume that $\langle f, e\rangle$ in S_2 since we can prove by induction that if $\langle c, e\rangle \in Q_3$, there must be some $x \in V$ such that $\langle c, x\rangle \in S_3$ and $\langle x, e\rangle \in S_2$.

Basis: If $\langle f, e\rangle$ is in S_2, we may set $x = f$ such that the hypothesis holds.

Step: If $\langle f, e\rangle$ is in not S_2, it must be in Q_3. By the induction hypothesis, there must be some $x \in V$ such that $\langle f, x\rangle \in S_3$ and $\langle x, e\rangle \in S_2$. This means that $\langle c, x\rangle$ must be in S_3 since both $\langle c, f\rangle$ and $\langle f, x\rangle$ are in S_3, and **part-of**(c, x) follows from **part-of**(c, f) and **part-of**(f, x) by option 3. Hence, the hypothesis holds.

We may therefore replace f by x if $\langle f, e\rangle$ is in not S_2. Hence, assume that $\langle f, e\rangle$ is in S_2. We have also that $\langle c, f\rangle \in S_3$ and **is-a**(d, e) holds. **part-of**(f, d) follows from **part-of**(f, e) and **is-a**(d, e) by option 2. So, $\langle f, d\rangle$ will be in S_2. Therefore, $\langle c, d\rangle$ is in S_3 since both $\langle c, f\rangle$ and $\langle f, d\rangle$ are in S_3. So, we have obtained a contradiction (since $\langle c, d\rangle$ was not supposed to be in S_3). Hence, Q_4 must be subset of S_3.

We still need to prove that $Q_j \subseteq S_3$ for $j \geq 5$. Since S_3 contains all of the tuples in Q_4, S_4 must be equal to S_3. S_3 is created by applying option 3 recursively on S_2. So, if S_4 contains no other tuples than those in S_3, no additional tuples will be made when option 3 is applied on S_4 in order to create S_5. This means that Q_5 will be a subset of S_3, and S_5 will be equal to S_3. The situation will be the same for Q_6 since this set will be created from S_3 just as Q_4. We may continue this deduction for any j. However, since neither option manages to add any tuple, Q_j must be a subset of S_3 for $j \geq 4$. Hence, **part-of** must be equal to S_3. □

Since Definition 6.3 is stratified, we may consider the two first options independently from the last option. For the two first options, we may derive this property.

Lemma 6.5. *If* $\textbf{\textit{part-of}}(c, d)$ *follows from option 2 and 1 (but not option 3) in Definition 6.3, there must be some* $x \in V$ *where* $\langle c, x\rangle \in P$ *and* $\textbf{\textit{is-a}}(d, x)$*.* ■

Proof. Since $\textbf{part-of}(c, d)$ follows option 2, we know that $\textbf{part-of}(c, e)$ and $\textbf{is-a}(d, e)$ hold for some $e \in V$. We prove by induction that there must be some $x \in V$ where $\langle c, x\rangle \in P$ and $\textbf{is-a}(d, x)$:

Basis: If $\langle c, e\rangle \in P$, we may set $x = e$. Hence, the induction hypothesis holds.

Step: If $\langle c, e\rangle \notin P$, we may assume that $\textbf{part-of}(c, e)$ must be derived from option 2 since we do not consider option 3. By the induction hypothesis, there must be some $x \in V$ where $\langle c, x\rangle \in P$ and $\textbf{is-a}(e, x)$. Since $\textbf{is-a}(d, e)$, we have by the transitivity of **is-a** that $\textbf{is-a}(d, x)$. So the hypothesis holds also in this case. □

The following theorems prove that Algorithm 6.1 creates a new DAG with a derived relation $\textbf{slg}'$ that is equivalent to the **slg**-relation derived from of the original GO-DAG.

Theorem 6.1 (Correctness Algorithm 6.1). *Let* $D = \langle V, \top, I, P\rangle$ *be a GO-DAG, and assume that the algorithm is called as* $\textbf{\textit{Transform}}(D, \top, \emptyset)$*. Then it will produce a DAG* $D' = \langle V', \top', E'\rangle$*. The* $\textbf{\textit{slg}}'$*-relation derived from* D' *corresponds to the* $\textbf{\textit{slg}}$*-relation derived from* D *such that for all* $c, d \in V$ $\textbf{\textit{slg}}'(c, d)$ *implies* $\textbf{\textit{slg}}(c, d)$*.* ■

Proof. If $\textbf{slg}'(c, d)$ holds, one of the two options in Definition 6.5 must hold. We prove by induction on this definition that $\textbf{slg}'(c, d)$ implies $\textbf{slg}(c, d)$.

Basis: Assume that $\langle c, d\rangle \in E'$. The edges for d are added to E' in only line 11 and 14 when the algorithm calls itself recursively as $\textbf{Transform}(D, d, L)$. In line 11, $c \in C_I(d)$ so that $\langle c, d\rangle \in I$. Hence, $\textbf{slg}(c, d)$ follows from Definition 6.2 and 6.4. In line 14, c may belong to L or L'. If $c \in L'$ then $c \in C_P(d)$ and $\langle c, d\rangle \in P$. Thus, $\textbf{slg}(c, d)$ follows from Definition 6.3 and 6.4. If $c \in L$, $\textbf{part-of}(c, d)$ holds by Lemma 6.2, and $\textbf{slg}(c, d)$ follows by Definition 6.4.

Step: If $\textbf{slg}'(c, d)$ is derived from the transitive closure in Definition 6.5, we may select an $e \in V$ such that $\textbf{slg}'(c, e)$ and $\langle e, d\rangle \in E'$. We have just proven that $\textbf{slg}(c', d')$ holds for any edge $\langle c', d'\rangle \in E'$. Hence, $\langle e, d\rangle \in E'$ implies $\textbf{slg}(e, d)$. Furthermore, $\textbf{slg}'(c, e)$ implies $\textbf{slg}(c, e)$ by the induction hypothesis. Then $\textbf{slg}(c, d)$ follows from the transitivity of **slg** and $\textbf{slg}(c, e)$ and $\textbf{slg}(e, d)$. □

Theorem 6.2 (Completeness of Algorithm 6.1). *Given the same conditions as in Theorem 6.1. The* $\textbf{\textit{slg}}$*-relation derived from* D *corresponds to the* $\textbf{\textit{slg}}'$*-relation derived from* D' *such that for all* $c, d \in V$ $\textbf{\textit{slg}}(c, d)$ *implies* $\textbf{\textit{slg}}'(c, d)$*.* ■

Proof. **slg**(c, d) may be derived in several different ways from Definitions 6.2, 6.3, and 6.4. For each we prove that **slg**$'(c, d)$ must follow:

1. If **slg**(c, d) is derived from option 1 in Definition 6.4, we have that **is-a**(c, d) and may prove by induction on Definition 6.2 that **slg**$'(c, d)$ follows.

   Basis: If $\langle c, d\rangle \in I$ then $\langle c, d\rangle$ is inserted into E' in line 11, and **slg**$'(c, d)$ follows.

   Step: If **is-a**(c, d) is derived from option 2 in Definition 6.2, we may select an e such that **is-a**(c, e) and $\langle e, d\rangle \in I$, and as before **slg**$'(e, d)$ follows. By the induction hypothesis **slg**$'(c, e)$ must follow from **is-a**(c, e). Then **slg**$'(c, d)$ holds by the transitivity of **slg**$'$.

2. If **slg**(c, d) is derived from option 2 in Definition 6.4, **part-of**(c, d) must hold, and just as for **is-a**, we may prove that **slg**$'(c, d)$ follows by induction on Definition 6.3. However, we have two recursive options in this case. We therefore apply Lemma 6.4 and consider option 2 as a basis case with respect to option 3.

   Basis:

   (a) If $\langle c, d\rangle \in P$, the algorithm will take two different actions depending on contents of $C_I(d)$:

      i. If $C_I(d) = \emptyset$, $\langle c, d\rangle$ is inserted into E' in line 14, and **slg**$'(c, d)$ follows.

      ii. If $C_I(d) \neq \emptyset$, c will be in L' the first d is visited since $L' = C_P(d)$ and $c \in C_P(d)$. **Transform**$(D, x, L \cup L')$ will be called for all $x \in C_I(d)$ in line 17. This means that **slg**$'(c, x)$ must hold for all $x \in C_I(d)$ by Lemma 6.3. Moreover, each tuple $\langle x, d\rangle \in I$ is inserted in E' line 11 such that **slg**$'(x, d)$ holds for all $x \in C_I(d)$. By the transitivity of **slg**$'$, **slg**$'(c, d)$ follows.

   (b) If **part-of**(c, d) follows from option 2 in Definition 6.3, there must be some $x \in V$ where $\langle c, x\rangle \in P$ and **is-a**(d, x) by Lemma 6.5. This means that c must be in L' the first time x is visited since $c \in C_P(x)$. Since **is-a**(d, x) holds, there must be an **is-a**-path from d to x. In this case, we may separate between two cases.

      i. If $\langle d, x\rangle \in I$, it follows from the first part of Lemma 6.3 that **slg**$'(c, d)$ because $c \in L'$ when **Transform**$(D, d, L \cup L')$ is called in line 17.

      ii. If $\langle d, x\rangle \notin I$, there must be some $y \in V$ such that $\langle y, x\rangle \in I$ and **is-a**(d, y). c will be in L', when y is called with **Transform**$(D, y, L \cup L')$. Hence, the second part of Lemma 6.3 may be applied. This states that **slg**$'(c, z)$ must hold for all $z \in V$ where **is-a**(z, y). This means **slg**$'(c, d)$ must be true since **is-a**(d, y) holds (set $z = d$).

   Step: If **part-of**(c, d) is derived from option 3 in Definition 6.3, we may prove **slg**$'(c, d)$ in the same way as we did for **is-a**.

3. If **slg**(c, d) is derived from option 3 in Definition 6.4, we may prove by induction that **slg**$'(c, d)$ must hold. The proofs for **is-a** and **part-of** constitute the basis in this case, and the proof of the step is analogous to the proofs of step for **is-a** and **part-of**. □

6.8 Some remarks about research methodology

We have now shown that the gene ontology can be transformed into a DCA-compliant DAG. Annotations in the GO can therefore be predicted by a learning algorithm that considers only a DAG of this kind. We will consequently assume that the DAG is DCA-compliant when we develop learning algorithms in the next chapters.

However, it is not obvious that the transformation algorithm should be applied on the ontology, which is currently delivered by the GO Consortium. The gene ontology is still under development and is immature. It is unclear whether the designers of the GO have considered the consequences that the inheritance rule has on the True Path Rule and whether the rule is applied by the people who perform the annotation work. There are several ambiguities in the ontology, and the semantics given by the GO Consortium suggests that the rule has not been implemented yet.

First, the definition of the True Path Rule is very informal[6] and seems to ignore Rule 3. It states only that a gene annotated to a concept belongs to the parents. The definition obeys consequently the DCA and the DAG may be used directly. However, this is mainly due to lack of formal analysis of the semantics since it makes little sense to introduce additional **part-of**-relationships by inheritance and then ignore these relationships when the annotations are considered. The principle behind the True Path Rule is after all that any gene annotated to a concept is also annotated to a more general concept. So if we assume that the **part-of**-relationships defined by inheritance are valid, we must assume that the ordering of the genes corresponds to these relationships. The GO Consortium's informal True Path Rule is thus flawed. However, since the annotations created by them are supposedly based on this rule, it may be better at this stage to neglect the inheritance rule.

Second, it seems that inheritance is not properly modeled in the ontology. One example is the subhierarchy under *Protein-nucleus import* shown in Figure 6.6. Rule 3 introduces the dashed lines in this figure. According to the rule, *Protein-nucleus import, docking* is a part of *ribosomal protein-nucleus import* and *NF-κB protein-nucleus import*. This is biologically correct as docking is a part of these processes. However, the genes in *Protein-nucleus import, docking* are not necessarily a subset of genes in *ribosomal protein-nucleus import*. *Protein-nucleus import, docking* is a quite general process. The docking procedure that is involved in an instance of *protein-nucleus import* is really specific to that particular instance. *Ribosomal protein-nucleus import*

[6]The formal definition given in Figure 6.4 was created by the author. However, the other rules (including Rule 3) are defined by the GO Consortium.

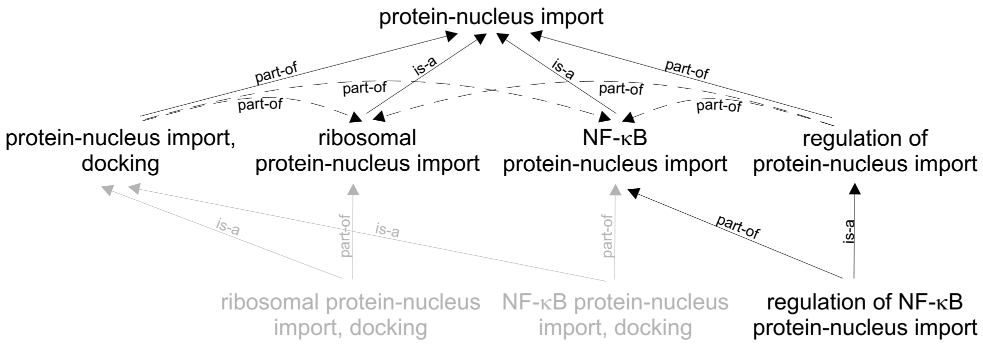

protein-nucleus import
part-of
is-a
is-a
part-of
part-of
part-of
part-of
part-of
protein-nucleus import, docking
ribosomal protein-nucleus import
NF-κB protein-nucleus import
regulation of protein-nucleus import
is-a
part-of
is-a
part-of
part-of
is-a
ribosomal protein-nucleus import, docking
NF-κB protein-nucleus import, docking
regulation of NF-κB protein-nucleus import

Figure 6.6: *Difficulties in the current ontology when applying inheritance.* Potential concepts and relations that do not exist in the current ontology are shown in gray. The dashed lines correspond to relations introduced by Rule 3.

should therefore have a docking process like ***ribosomal*** *protein-nucleus import, docking*, and *NF-κB protein-nucleus import* should have a process like ***NF-κB*** *protein-nucleus import, docking* (as shown in gray). These docking processes may contain different genes. However, they will be instances of *Protein-nucleus import, docking* so that this process will contain the genes of both processes. This means that the genes of *Protein-nucleus import, docking* may not be a subset of genes in *ribosomal protein-nucleus import* (and *NF-κB protein-nucleus import*). So, Rule 3 will produce an incorrect ordering in this case and should not be applied.

Moreover, *NF-κB protein-nucleus import* has a part called *regulation of NF-κB protein-nucleus import*. This part is also an instance of *regulation of protein-nucleus import*, which again is a part of *Protein-nucleus import*. This means that *regulation of protein-nucleus import* is a part of *NF-κB protein-nucleus import* by Rule 3. *NF-κB protein-nucleus import* has therefore two regulation processes where one is more general than the other. In this case, it seems that only the most specific concept should be a part of *NF-κB protein-nucleus import*. The rule should thus not be applied or the ontology should be changed so that *regulation of protein-nucleus import* is not a part of *Protein-nucleus import*.

Hence, there is evidence that Rule 3 is not currently implemented, and it appears that we may be better off by neglecting the rule for now. It was decided to not apply the transform in the experiments in Chapter 10, but rather assume that the Gene Ontology is DCA-compliant.

The transformation algorithm may therefore seem unnecessary. However, it is still important. It demonstrates that the methods, which will be introduced in the next chapters, may be used on an ontology that implements the inheritance rule. So, when the GO becomes more mature, we may apply the transform and use these methods on the transformed ontology. Furthermore, inheritance is a well-known concept and may be required under other circumstances. The transform shows that the methods may be applied in these situations as well.

A Rough Set Framework for the DAG 7

Parts of this chapter have previously been presented in [113].

7.1 Introduction

In the last chapter, we saw that the function of genes may be described by classes that appear in a DAG. The DAG defines relationships between the classes. A learning algorithm, which tries to predict the function of a gene, must thus consider these relationships. However, ordinary learning algorithms cannot take such relationships into account. DAG learning algorithms are therefore needed and will be introduced in the next chapter.

In order to develop such algorithms, we need a formal framework that can handle the uncertainty about the data just as in ordinary learning problems. In our case, we would like to use rough set theory, but rough set theory cannot be used directly on this learning problem. A decision system assumes that the decision classes constitute a flat set and cannot represent the DAG. This means that the upper and the lower approximations cannot be used without modifications.

A DAG-decision system is therefore introduce in this chapter. The ambiguities that arise in the DAG are also discussed, and several approximations for the DAG-decision system are defined.

7.2 A *more general* order on the DAG

As in Section 6.7, we begin by defining an ordering relation on the DAG. The following relation is DCA-compliant.

Definition 7.1 (More general). Let $G = \langle V_d, E \rangle$ be a DAG where V_d is a set of decision classes and $E \subseteq V_d \times V_d$ is a set of edges in the graph. The relation $c \succcurlyeq e$ denotes that c is *more general* than e ($c, e \in V_d$) and is defined on G such that

1. $c \succcurlyeq c$ for all $c \in V_d$
2. $c \succcurlyeq e$ for all $\langle e, c\rangle \in E$[1].
3. for all $c, f \in V_d$, $c \succcurlyeq f$, if there is an $e \in V_d$ such that $c \succcurlyeq e$ and $e \succcurlyeq f$. ■

Definition 7.2 (Less general). $c \preccurlyeq e$ denotes that c is *less general* than e and holds iff $e \succcurlyeq c$ ■

$\succcurlyeq$ (and $\preccurlyeq$) is a partial order as stated by the following proposition.

Proposition 7.1. *$\succcurlyeq$ is a partial order on V_d.* ■

Proof. $\succcurlyeq$ is reflexive and transitive by cases 1 and 3, respectively, and it remains to show that $\succcurlyeq$ is anti-symmetric. Assume that $\succcurlyeq$ is not anti-symmetric. Then, there are some classes x and y in V_d such that $x \succcurlyeq y$ and $y \succcurlyeq x$, but $x \neq y$. $x \succcurlyeq y$, together with cases 2 and 3, implies that there is a path $\langle y, v_1, \ldots, v_k, x\rangle$ from y to x in G. Similarly, there must be a path $\langle x, w_1, \ldots, w_l, y\rangle$ from x to y (since $y \succcurlyeq x$). Then, there is a cycle $\langle x, w_1, \ldots, w_k, y, v_1, \ldots, v_l, x\rangle$ in the graph, and this results in a contradiction. Hence, $\succcurlyeq$ is anti-symmetric if G is acyclic. This means that $\succcurlyeq$ must be a partial order. □

The following relations can be derived from $\succcurlyeq$.

Definition 7.3 (Strictly more general). Given a partial order $\succcurlyeq$ on a set of classes V_d, we say that c is *strictly more general* than e ($c, e \in V_d$), denoted as $c \succ e$, iff $c \succcurlyeq e$ and $c \neq e$. ■

Definition 7.4 (Related classes). Let $\succcurlyeq$ be a partial order on a set of classes V_d. The classes $c, e \in V_d$ are *related*, denoted as $c \approx e$, iff $c \succcurlyeq e$ or $e \succcurlyeq c$. If $c \not\approx e$, we say that c and e are *unrelated*. ■

Notice that $\approx$ is reflexive and symmetric, but not transitive. The classes, which are above, below, or related to a class c, are denoted by these sets:

- **Above set**: $[c]^{\succcurlyeq} = \{e \in V_d \mid e \succcurlyeq c\}$
- **Below set**: $[c]^{\preccurlyeq} = \{e \in V_d \mid c \succcurlyeq e\}$
- **Related set**: $[c]^{\approx} = \{e \in V_d \mid c \approx e\}$

In the following, we will also need these definitions.

Definition 7.5 (Classes).

- A class c is a *superclass* of e if $c \succ e$. A class c is an *immediate* superclass of e if it is a superclass of e and there is no $f \in V_d$ such that $c \succ f \succ e$.

[1]As in Section 6.7, we assume that an edge points from the most specific to the most general class. So, c is more general than e in this case.

- A class c is a *subclass* of e if $e \succ c$. A class c is an *immediate* subclass of e if it is a subclass of e and there is no $f \in V_d$ such that $e \succ f \succ c$.
- A *leaf* class c has no subclasses. Hence, for all $e \in V_d$, $c \not\succ e$.
- A *non-leaf* class c has at least one subclass. Hence, there is some $e \in V_d$ such that $c \succ e$.
- A *root* class c has no superclasses. Hence, for all $e \in V_d$, $e \not\succ c$ holds for c. ■

For a class c, the set of its immediate superclasses ($Sup(c)$) and the set of its immediate subclasses ($Sub(c)$) are denoted respectively as follows:

$$Sup(c) = \{e \in V_d \mid e \succ c \text{ and there is no } f \in V_d \text{ such that } e \succ f \succ c\}$$
$$Sub(c) = \{e \in V_d \mid c \succ e \text{ and there is no } f \in V_d \text{ such that } c \succ f \succ e\}$$

7.3 Paths

The concept *path* can be defined in two different ways. In Definition A.13, a path is defined directly on edges of the DAG. Here, we will define it on the partial order $\succcurlyeq$ (and $\preccurlyeq$).

Definition 7.6 (Path). $\langle c_0, c_1, \ldots, c_n \rangle$ is *path* from class c_0 to class c_n if $c_0 \prec c_1 \prec \ldots \prec c_n$. ■

In the following, we will use this definition since we will only consider the partial order $\succcurlyeq$ (and $\preccurlyeq$).

By the length of a path, we mean the number of edges on the path. This number is one less than the number of classes in the path.

Definition 7.7 (Length of a path). Let $t = \langle c_0, c_1, \ldots, c_n \rangle$ be a path. Then the length of a path t, $||t|| = n$ (i.e., $||t||$ is equal to the number of edges on the path). ■

There may be many paths between two classes. We denote the set of paths from class a to class b as $Paths(a, b)$.

Definition 7.8 (Paths). The set of paths from a to b ($b \succ a$), is

$$Paths(a, b) = \{\langle c_0, c_1, \ldots, c_n \rangle \mid a = c_0 \prec c_1 \prec \ldots \prec c_n = b\}$$ ■

In Chapters 8 and 9, we will need to perform some simple operations on paths. $s \sqsubseteq t$ denotes that s is a subpath t, and $e \lhd t$ states that path t contains the class e.

Definition 7.9 (Subpath). Let $s = \langle b_0, b_1, \ldots, b_m \rangle$ and $t = \langle c_0, c_1, \ldots, c_n \rangle$ be two paths. s is a subpath of t, denoted as $s \sqsubseteq t$, if there is mapping such that $b_i = c_{i+j}$ for all $0 \leq i \leq m$ where $0 \leq j \leq n - m$. ■

Definition 7.10 (Path membership). Let $t = \langle c_0, c_1, \ldots, c_n \rangle$ be a path from c_0 to c_n. $e \lhd t$ denotes that there is some i ($0 \leq i \leq n$) such that $e = c_i$. ■

7.4 DAG-decision systems

A DAG-decision system is an extension of a decision system and is defined as follows.

Definition 7.11 (DAG-decision system). Let $\mathcal{A} = \langle U, A, d, \succcurlyeq \rangle$ denote a DAG-decision system where

- U is a non-empty finite set of (observable) objects, called the universe.
- A is a set of conditional attributes describing the objects. Each attribute $a \in A$ is a function $a : U \to V_a$ where V_a is a set of values that an object may have for a.
- d is the decision attribute, which is not in A. It is a function $d : U \to V_d$ where V_d is a set of decision classes.
- $\succcurlyeq$ is a partial order on the classes in V_d where $p \succcurlyeq r$ denotes that p is *more general than* r $(p, r \in V_d)$. ■

Only one decision class is assigned to each object in this system. A gene may, however, have several annotations. In order to reflect this, we also introduce a DAG-decision system with multiple decisions for each object.

Definition 7.12 (DAG-decision system with multiple decision classes). Let $\mathcal{A} = \langle U, A, D, \succcurlyeq \rangle$ denote a DAG-decision system with multiple decision classes per object where U, A, and $\succcurlyeq$ are as defined in Definition 7.11. The decision attribute D, is a function $D : U \to \mathcal{P}(V_d)$ where $\mathcal{P}(V_d)$ is the power set on the decision classes in V_d. ■

The upper and the lower approximations cannot be applied directly to a DAG-decision system with multiple decisions. Hence, it seems that we would have to define new approximations for this system. Fortunately, this is not necessary. The decision system can be transformed into an ordinary DAG-decision system with unique decisions, and the standard approximations may be applied to the transformed system. So, we will not introduce any approximations, but provide a transformation instead.

Definition 7.13 (Transformation). A DAG-decision system with multiple classes per object $\mathcal{A} = \langle U, A, D, \succcurlyeq \rangle$ may be transformed into a DAG-decision system with a unique class per object $\mathcal{A}' = \langle U', A', d', \succcurlyeq \rangle$ as follows:

- $U' = \{\langle x, d \rangle \mid x \in U, d \in D(x)\}$
- $A' = \{a' \mid a \in A\}$ where a' is defined as $a'(\langle x, d \rangle) = a(x)$, for each $\langle x, d \rangle \in U'$
- $d'(\langle x, d \rangle) = d$, for each $\langle x, d \rangle \in U'$ ■

Note that the objects (genes) in the original system and their associated set of classes may be recreated by an indiscernibility relation that partitions the universe U' (in the transformed system) according to the objects in the original system.

Definition 7.14. Let $\mathcal{A}' = \langle U', A', d', \succcurlyeq \rangle$ be a transformed DAG-decision system. The indiscernibility relation *IND*(G) partitions the universe into a collection of elementary sets that correspond to the objects in the original system:

$$IND(G) = \{\langle x, y\rangle \in U' \times U' \mid x = \langle z, d_x\rangle \text{ and } y = \langle z, d_y\rangle\} \qquad \blacksquare$$

The generalized decision defined on *IND*(G) is:

$$\partial_G(x) = \{d'(y) \mid y \in [x]_G\}$$

This is similar to the original set of classes so that $D(x) = \partial_G(\langle x, d_x\rangle)$ for any $\langle x, d_x\rangle \in U'$. Moreover, *IND*$(G)$ creates a finer partition than *IND*(A'). Hence, we have that $\overline{A'}X = \overline{A'}\,\overline{G}X$ and $\underline{A'}X = \underline{A'}\,\underline{G}X$.

Theorem 7.1.

a) $\overline{A'}X = \overline{A'}\,\overline{G}X$

b) $\underline{A'}X = \underline{A'}\,\underline{G}X$ $\blacksquare$

Proof. Note that $y \in [x]_A$ iff $[y]_G \subseteq [x]_A$. This follows directly from the fact that every object in $[y]_G$ is created from the same object in the original DAG-decision system and has therefore the same information vector.

a) $\overline{A'}\,\overline{G}X = \{x \in U' \mid [x]_A \cap \overline{G}X \neq \emptyset\} = \{x \in U' \mid y \in [x]_A \text{ and } y \in \overline{G}X\}$
$= \{x \in U' \mid y \in [x]_A \text{ and } z \in [y]_G \text{ and } z \in X\}$
Since $y \in [x]_A$ iff $[y]_G \subseteq [x]_A$, we have
$= \{x \in U' \mid [y]_G \subseteq [x]_A \text{ and } z \in [y]_G \text{ and } z \in X\}$
$= \{x \in U' \mid z \in [x]_A \text{ and } z \in X\} = \{x \in U' \mid [x]_A \cap X \neq \emptyset\} = \overline{A'}X$

b) $\underline{A'}\,\underline{G}X = \{x \in U' \mid [x]_A \cap \overline{G}X \neq \emptyset\} = \{x \in U' \mid \forall y\ y \in [x]_A \to y \in \overline{G}X\}$
$= \{x \in U' \mid \forall y \forall z\ y \in [x]_A \to (z \in [y]_G \to z \in X)\}$
$= \{x \in U' \mid \forall z\ (\exists y\ y \in [x]_A \wedge z \in [y]_G) \to z \in X)\}$
From the property $y \in [x]_A$ iff $[y]_G \subseteq [x]_A$, it follows that
$= \{x \in U' \mid \forall z\ (\exists y\ [y]_G \subseteq [x]_A \wedge z \in [y]_G) \to z \in X)\}$
$= \{x \in U' \mid \forall z\ z \in [x]_A \to z \in X)\} = \{x \in U' \mid [x]_A \subseteq X\} = \underline{A'}X$ $\square$

In the following, we will assume (unless it is explicitly stated) that the transform has been applied such that the DAG-decision system has only unique decisions.

7.5 The complement of a class

The complement of a class c is the set of classes that should be separated from c by a learning algorithm. In an ordinary (flat) decision system, we want to separate each class from the other classes in V_d. A learning algorithm will thus try to discern the objects of c (i.e., the objects in X_c) from the objects of $V_d - \{c\}$ (i.e., the objects in $\bigcup_{e \in V_d - \{c\}} X_e = U - X_c$). Hence, the complement of c may be defined as $V_d - \{c\}$ in this case.

This definition is not appropriate for a DAG-decision system. The DAG defines relationships between the classes, and these relationships create implicit annotations. For example, an object does not only belong to the class(es) to which it is labeled, it also belongs to the superclasses of the class(es). Thus, we define the complement of c to be the set of classes that are unrelated to c.

Definition 7.15 (Complement). The complement of a class c is the set of classes that are unrelated to c:

$$\sim c = \{e \in V_d \mid e \not\approx c\}$$

For a set of classes $C \subseteq V_d$, the complement, denoted by $\sim C$, is the set of classes that are unrelated to any class in C:

$$\sim C = \{e \in V_d \mid \text{for all } c \in C,\ e \not\approx c\} \qquad \blacksquare$$

Note that $\sim\{c\}$ is same as the single class complement $\sim c$. The complement $\sim C$ is equal to the intersection of the single class complements.

Lemma 7.1. $\sim C = \left(\bigcap_{c \in C} \sim c\right)$ *if we assume that* $\left(\bigcap_{c \in C} \sim c\right) = V_d$ *when* $C = \emptyset$. $\blacksquare$

This is seen easily. So no proof is given. The complement has the following properties on unions and intersections of classes.

Lemma 7.2. $\sim(C \cup D) = \sim C \cap \sim D$ $\blacksquare$

Proof. Follows directly from Lemma 7.1. $\square$

Lemma 7.3. $\sim(C \cap D) \supseteq \sim C \cup \sim D$ $\blacksquare$

Proof. According to Lemma A.4, we have that $A \cap B \subseteq A$ holds for any sets A and B. So, $\sim C \subseteq \sim(C \cap D)$ since

$$\sim C = \bigcap_{c \in C} \sim c = \left(\bigcap_{c \in (C \cap D)} \sim c \right) \cap \left(\bigcap_{c \in (C - D)} \sim c \right) \subseteq \bigcap_{c \in (C \cap D)} \sim c = \sim(C \cap D)$$

Similarly, $\sim D \subseteq \sim(C \cap D)$. Then $\sim C \cup \sim D \subseteq \sim(C \cap D)$ holds since for any sets A, A', and B, we have that $A \subseteq B$ and $A' \subseteq B$ implies $A \cup A' \subseteq B$ by Lemma A.5. $\square$

Unfortunately, $C \neq \sim(\sim C)$. For example, let $C = \{c\}$ in Figure 7.1, then $\sim C = \{b_1, b_2, b_3\}$ and $\sim(\sim C) = \{c, d_1\}$. C is nonetheless a subset of $\sim(\sim C)$.

Lemma 7.4. $C \subseteq \sim(\sim C)$ ■

Proof. Assume false. Then, there must be an $x \in C$, which is not in $\sim(\sim C)$. Since $x \notin \sim(\sim C)$, there must be a $y \in \sim C$ such that $x \approx y$. However, if $y \in \sim C$ implies that $x \not\approx y$ since $x \in C$. So, we obtain a contradiction. □

Even though $\sim(\sim C) \subseteq C$ does not hold, we may prove that $\sim(\sim(\sim C)) \subseteq \sim C$.

Lemma 7.5. $\sim C = \sim(\sim(\sim C))$ ■

Proof. $\sim C \subseteq \sim(\sim(\sim C))$ follows from Lemma 7.4. Therefore, consider the statement $\sim(\sim(\sim C)) \subseteq \sim C$. Assume that it does not hold so that there is an $x \in \sim(\sim(\sim C))$, but $x \notin \sim C$. Then x must be related to some $c' \in C$ since $x \notin \sim C$. This c' must also be in $\sim(\sim C)$ since $C \subseteq \sim(\sim C)$ (by Lemma 7.4). Then we have that $x \approx c'$ holds for some $c' \in \sim(\sim C)$. This means that x cannot be in $\sim(\sim(\sim C))$. Hence, we have obtained a contradiction so that $\sim(\sim(\sim C)) \subseteq \sim C$ must be true. □

We may also prove a weak subset relation between $\sim(\sim C)$ and C.

Lemma 7.6. *For all $e \in V_d$, if there is some $c' \in \sim(\sim C)$, such that $e \approx c'$ and $c' \in \sim(\sim C)$ then there is also some $c \in C$ such that $e \approx c$.* ■

Proof. Choose an arbitrary $e \in V_d$. If $c' \in \sim(\sim C)$ and $e \approx c'$, we know that $e \notin \sim C$ since c' cannot be related to any $c'' \in \sim C$ according to Definition 7.15. Since $e \notin \sim C$, there must be some $c \in C$ such that $e \approx c$. □

Thus, the set of classes related to the classes in $\sim(\sim C)$ is a subset of the classes related to the classes in C. This lemma allows us to derive several important lemmas that will be needed in the next sections. In particular, we may prove following properties on C and $\sim(\sim C)$.

Lemma 7.7.

a) $[e]^{\approx} \cap \sim(\sim C) \neq \emptyset \Leftrightarrow [e]^{\approx} \cap C \neq \emptyset$

b) $[e]^{\precsim} \cap C \neq \emptyset \Rightarrow [e]^{\approx} \cap C \neq \emptyset$

c) $[e]^{\succsim} \cap C \neq \emptyset \Rightarrow [e]^{\approx} \cap C \neq \emptyset$

d) $[e]^{\precsim} \cap C \neq \emptyset \Rightarrow [e]^{\precsim} \cap \sim(\sim C) \neq \emptyset$

e) $[e]^{\succsim} \cap C \neq \emptyset \Rightarrow [e]^{\succsim} \cap \sim(\sim C) \neq \emptyset$ ■

Proof.

a) The statement that $[e]^{\approx} \cap \sim(\sim C) \neq \emptyset$ implies $[e]^{\approx} \cap C \neq \emptyset$ is a direct consequence of Lemma 7.6. The opposite statement that $[e]^{\approx} \cap C \neq \emptyset$ implies $[e]^{\approx} \cap \sim(\sim C) \neq \emptyset$ follows from Lemma A.1 since $C \subseteq \sim(\sim C)$.

b) Follows from Lemma A.1 since $[e]^{\preccurlyeq} \subseteq [e]^{\approx}$.

c) Follows from Lemma A.1 since $[e]^{\succcurlyeq} \subseteq [e]^{\approx}$.

d) Follows from Lemma A.1 since $C \subseteq {\sim}({\sim}C)$ by Lemma 7.4.

e) Follows from Lemma A.1 since $C \subseteq {\sim}({\sim}C)$ by Lemma 7.4. □

The above set $[e]^{\succcurlyeq}$ and the below set $[e]^{\preccurlyeq}$ of a class e are related through the complement such that if some class in ${\sim}({\sim}C)$ is in the above set (below set) then no class in ${\sim}C$ is in the below set (above set), and vice versa. Note this is due to a special property of the complement and does not hold for C.

Lemma 7.8. *For all $e \in V_d$, if $x \in [e]^{\succcurlyeq}$ and $y \in [e]^{\preccurlyeq}$ then $x \succcurlyeq y$.* ■

Proof. $x \in [e]^{\succcurlyeq}$ implies $x \succcurlyeq e$, and $y \in [e]^{\preccurlyeq}$ implies $e \succcurlyeq y$. Then, $x \succcurlyeq y$ by transitivity. □

Lemma 7.9. $[e]^{\preccurlyeq} \cap {\sim}({\sim}C) \neq \emptyset$ *iff* $[e]^{\succcurlyeq} \cap {\sim}C = \emptyset$ ■

Proof.

⇒: There must be some y in $[e]^{\preccurlyeq} \cap {\sim}({\sim}C)$ since this set is non-empty. Then $y \in [e]^{\preccurlyeq}$ and $y \in {\sim}({\sim}C)$. By Lemma 7.8, $x \succcurlyeq y$ holds for any $x \in [e]^{\succcurlyeq}$. Since $y \in {\sim}({\sim}C)$ and $x \approx y$, x must be related to some $c \in C$ by Lemma 7.6. As this holds for any $x \in [e]^{\succcurlyeq}$, no $x \in [e]^{\succcurlyeq}$ may be in ${\sim}C$. Hence, $[e]^{\succcurlyeq} \cap {\sim}C$ must be empty.

⇐: No $y \in [e]^{\succcurlyeq}$ may be in ${\sim}C$ since $[e]^{\succcurlyeq} \cap {\sim}C = \emptyset$. This means that for every $y \in [e]^{\succcurlyeq}$ there is some $c \in C$ such that $y \approx c$. In particular, $e \in [e]^{\succcurlyeq}$ such that $e \approx c'$ holds for some $c' \in C$. Then, $e \succcurlyeq c'$ or $c' \succcurlyeq e$ must hold.

1. If $e \succcurlyeq c'$, we have that $c' \in [e]^{\preccurlyeq}$ so that $[e]^{\preccurlyeq} \cap C \neq \emptyset$ holds. Then, $[e]^{\preccurlyeq} \cap {\sim}({\sim}C) \neq \emptyset$ follows from $[e]^{\preccurlyeq} \cap C \neq \emptyset$ by Lemma 7.7d.
2. If $c' \succcurlyeq e$, we show that $e \in {\sim}({\sim}C)$ such that $[e]^{\preccurlyeq} \cap {\sim}({\sim}C) \neq \emptyset$. If $e \notin {\sim}({\sim}C)$ then there must be some $z \in {\sim}C$ such that $z \approx e$.
   If $z \succcurlyeq e$, $z \in [e]^{\succcurlyeq}$ and $[e]^{\succcurlyeq} \cap {\sim}C$ would not be empty. Contradiction.
   If $e \succcurlyeq z$, we have $c' \succcurlyeq e \succcurlyeq z$ and z cannot be in ${\sim}C$. Contradiction. □

Lemma 7.10. $[e]^{\succcurlyeq} \cap {\sim}({\sim}C) \neq \emptyset$ *iff* $[e]^{\preccurlyeq} \cap {\sim}C = \emptyset$ ■

Proof.

⇒: Since $[e]^{\succcurlyeq} \cap {\sim}({\sim}C) \neq \emptyset$, there must be some y such that $y \in [e]^{\succcurlyeq}$ and $y \in {\sim}({\sim}C)$. $y \succcurlyeq x$ holds for any $x \in [e]^{\preccurlyeq}$ by Lemma 7.8. Since $y \in {\sim}({\sim}C)$ and $x \approx y$, x must be related to some $c \in C$ by Lemma 7.6. Thus, no $x \in [e]^{\preccurlyeq}$ may be in ${\sim}C$, and $[e]^{\preccurlyeq} \cap {\sim}C$ must be empty.

$\Leftarrow$: No $y \in [e]^{\preccurlyeq}$ may be in $\sim C$ since $[e]^{\preccurlyeq} \cap \sim C = \emptyset$. This means that for every $y \in [e]^{\preccurlyeq}$ there is some $c \in C$ such that $y \approx c$. In particular, $e \in [e]^{\preccurlyeq}$ such that $e \approx c'$ holds for some $c' \in C$. Then, $e \succcurlyeq c'$ or $c' \succcurlyeq e$ must hold.

1. If $c' \succcurlyeq e$, we have that $c' \in [e]^{\succcurlyeq}$ so that $[e]^{\succcurlyeq} \cap C \neq \emptyset$ holds. Then, $[e]^{\succcurlyeq} \cap \sim(\sim C) \neq \emptyset$ follows from $[e]^{\succcurlyeq} \cap C \neq \emptyset$ by Lemma 7.7e.
2. If $e \succcurlyeq c'$, we show that $e \in \sim(\sim C)$ such that $[e]^{\succcurlyeq} \cap \sim(\sim C) \neq \emptyset$. If $e \notin \sim(\sim C)$ then there must be some $z \in \sim C$ such that $z \approx e$.

   If $e \succcurlyeq z$, $z \in [e]^{\preccurlyeq}$ and $[e]^{\preccurlyeq} \cap \sim C$ would not be empty. Contradiction.

   If $z \succcurlyeq e$, we have $z \succcurlyeq e \succcurlyeq c'$ and z cannot be in $\sim C$. Contradiction. □

7.6 Ambiguities in the DAG

In set theory, the membership of all objects in the universe is assumed to be known. An object y is either a member of a set X or it is not. This is also the starting point in rough set theory, but here an elementary set may belong to both X and the complement $U - X$. In this case, we say that the membership is inconsistent. Hence, rough set theory distinguishes between three different ways that an element can be related to a set: *in*, *not in*, and *inconsistent*.

We may also recognize a fourth membership category; it may be *unknown* whether an object belongs to a set or not. This situation arises in the DAG as shown in Figure 7.1 for class c. The objects of the superclasses a and *root* may belong to c or some of the classes that are unrelated to c, i.e., b_1, b_2, and b_3. However, we do not know which class. The membership of these objects is in other words unknown with respect to c.

This means that a framework for a DAG-decision system must be able to represent objects whose memberships are unknown. Since rough set theory does not have such facilities, we must extend it. We do this by employing the same strategy as rough set theory and define a *lower* and an *upper* boundary for each class. The lower boundary K_c, which is also called the *known* set, basically consists of the objects that belong to the class c or to the subclasses of c. Note that the objects of the subclasses are included in this set since they are also members of c. The upper boundary P_c, which is also called the *potential* set, is a superset of K_c. It contains the objects of the superclasses of c in addition to the objects of K_c. Using these two sets, we can represent the unknown objects with regard to c as the difference $P_c - K_c$.

Formally, we could try to define the lower and the upper boundaries as follows:

$$\begin{aligned} K_c &= \{x \in U \mid c \succcurlyeq d(x)\} \qquad (= \{x \in U \mid [d(x)]^{\succcurlyeq} \cap \{c\} \neq \emptyset\}) \\ P_c &= \{x \in U \mid c \approx d(x)\} \qquad (= \{x \in U \mid [d(x)]^{\approx} \cap \{c\} \neq \emptyset\}) \end{aligned}$$

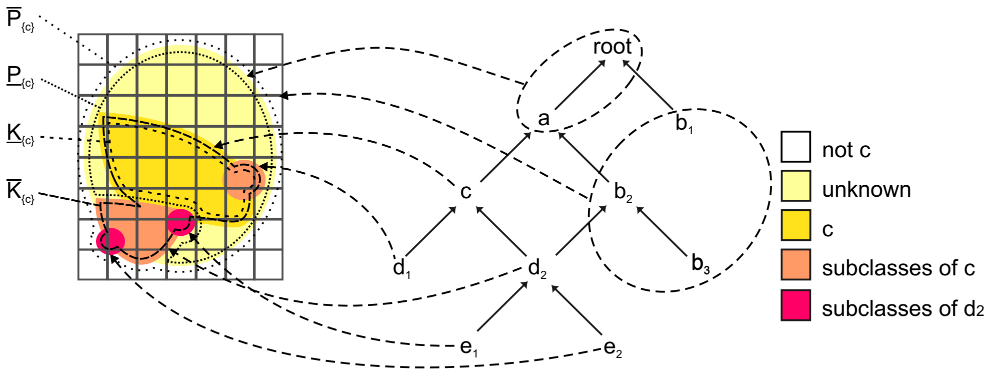


Figure 7.1: *Discernibility of the class c.*

For a set of classes $C \subseteq V_d$, these boundaries could also be defined as the union of the corresponding boundaries of each class, i.e.,

$$K_C = \bigcup_{c \in C} K_c = \{x \in U \mid [d(x)]^{\succsim} \cap \{c\} \neq \emptyset, \text{ for some } c \in C\} \text{ and}$$

$$P_C = \bigcup_{c \in C} P_c = \{x \in U \mid [d(x)]^{\approx} \cap \{c\} \neq \emptyset, \text{ for some } c \in C\}$$

Unfortunately, these definitions are complicated for several reasons.

One problem is that these boundaries are not complementary. We would like K_C to be complementary to $P_{\sim C}$ (i.e., $K_C = U - P_{\sim C}$) and P_C to be complementary to $K_{\sim C}$ (i.e., $P_C = U - K_{\sim C}$). Unfortunately, $K_C = U - P_{\sim C}$ does not hold.

Example 7.1. Assume that $C = \{d_1, d_2\}$ in Figure 7.1. Then K_C consists of the objects in d_1, d_2, e_1, and e_2. The complement $\sim C$ is, in this case, equal to $\{b_1, b_3\}$, and $P_{\sim C}$ consists of the objects in *root*, a, b_1, b_2, and b_3. However, the objects in c do not occur in either set. ■

The reason that the objects of the class c do not occur in K_C and $P_{\sim C}$ in the last example is that c has no other immediate subclasses than d_1 and d_2. All of its subclasses are related to C so that no class in the complement $\sim C$ is related to c. Its objects are therefore not included in $P_{\sim C}$. Moreover, $c \notin C$ so that its objects are not in K_C, either.

Notice that this issue is not specific to a DAG. It occurs in a tree as well (see Section 7.6.2). The problem also arises when a class has only a single child.

Example 7.2. Assume that the edge from d_2 to c is removed in Figure 7.1. In this case, d_1 is a lonely child of c. If $C = \{d_1\}$, then $\sim C = \{b_1, b_2, b_3, d_2, e_1, e_2\}$ and the objects of c are in neither K_C nor $P_{\sim C}$. ■

We may correct this problem by changing the definition of either K_C or $P_{\sim C}$. However, it would be counter-intuitive to define $P_{\sim C}$ so that it included the objects of c in Example 7.1. Class c is not a superclass of any class in $\sim C$, and its objects are not potential members of these classes. On the other hand, if we assume that the objects of c must belong to either d_1 or d_2, we *know* that the objects of c belong to C since C includes both classes. Hence, K_C should include objects from a class *if all of its subclasses are related to C*. More formally, object x should be in K_C if x satisfies

$$\forall e \in V_d \ (d(x) \succcurlyeq e \rightarrow \exists c \in C \ e \approx c)$$

This is equivalent to

$$\neg (\exists e \in V_d \ \ (d(x) \succcurlyeq e \wedge e \in \sim C))$$

which may be expressed as follows using the below set of $d(x)$

$$[d(x)]^{\preccurlyeq} \cap \sim C = \emptyset^2 \qquad \text{(Cond. 1)}$$

Note that this condition subsumes the condition in K_C (as shown by the following lemma). Thus, the objects in K_C will also satisfy Cond. 1, and we may replace the condition in the definition of K_C with Cond. 1.

Lemma 7.11. $[e]^{\succcurlyeq} \cap \{c\} \neq \emptyset$, *for some* $c \in C$ *implies* $[e]^{\preccurlyeq} \cap \sim C = \emptyset$ ■

Proof. $[e]^{\succcurlyeq} \cap \{c\} \neq \emptyset$, for some $c \in C$ is equivalent to $[e]^{\succcurlyeq} \cap C \neq \emptyset$ by Lemma A.2. $[e]^{\succcurlyeq} \cap C \neq \emptyset$ implies $[e]^{\succcurlyeq} \cap \sim(\sim C) \neq \emptyset$ by Lemma 7.7e. $[e]^{\succcurlyeq} \cap \sim(\sim C) \neq \emptyset$ is equivalent to $[e]^{\preccurlyeq} \cap \sim C = \emptyset$ by Lemma 7.10. □

There is another problem with the boundary sets. A class may have several immediate superclasses in a DAG. This means that some objects may be included in the (lower and upper) boundaries of both a set C and the complement $\sim C$.

Example 7.3. Consider Figure 7.1 again. If $C = \{c\}$, we have $\sim C = \{b_1, b_2, b_3\}$. The classes d_2, e_1, and e_2 are subclasses of c and b_2. Hence, they are below a class in C and a class in $\sim C$. Their objects will consequently be in both K_C and $K_{\sim C}$ (and P_C and $P_{\sim C}$). ■

The DAG introduces in this way a new type of inconsistency, which we will call DAG-inconsistency. Thus, we have to define upper and lower approximations of K_C and P_C. The upper approximations contain all objects from the subclasses of the classes in C, just as before. The lower approximations, on the other hand, do not contain the objects from subclasses, which introduce this kind of inconsistency.

A DAG-inconsistency occurs when a subclass of a class in C also has another superclass in $\sim C$. The subclass has in this case a superclass that is not related to any class in C. Hence, the objects of a class should only be included in the lower approximations if *all superclasses are related to a class in C*. Formally, object x should be included in the lower approximations if

$$\forall e \in V_d \ (e \succcurlyeq d(x) \rightarrow \exists c \in C \ e \approx c)$$

²In [113], we used the first condition that is stated in first-order logic. Here, we will use the set based condition instead as these simplify the proofs.

which is equivalent to

$$[d(x)]^{\curlyeqsucc} \cap {\sim}C = \emptyset \qquad \text{(Cond. 2)}$$

Note that it is necessary to define this condition for a set of classes since a DAG-inconsistency that occurs for a single class may disappear if a set is considered.

Example 7.4. Let $C' = \{c, b_2\}$ (in Figure 7.1), which is a superset of C in Example 7.3. In this case, ${\sim}C' = \{b_1\}$, and the objects of d_2, e_1, and e_2 belong only to $K_{C'}$ and $P_{C'}$ and not to $K_{\sim C'}$ and $P_{\sim C'}$. So the DAG-inconsistency that occurred for c in Example 7.3 has vanished. ■

We may define the following upper and lower approximations of the boundary sets:

- $\underline{K}_C = \{x \in U \mid \text{Cond. 1 and Cond. 2}\}$
- $\overline{K}_C = \{x \in U \mid \text{Cond. 1}\}$
- $\underline{P}_C = \{x \in U \mid [d(x)]^{\approx} \cap C \neq \emptyset \text{ and Cond. 2}\}$
- $\overline{P}_C = \{x \in U \mid [d(x)]^{\approx} \cap C \neq \emptyset\}$

Some simplifications are possible, however. The conditions in $\underline{K}_C$, $[e]^{\curlyeqprec} \cap {\sim}C = \emptyset$ (Cond. 1) and $[e]^{\curlyeqsucc} \cap {\sim}C = \emptyset$ (Cond. 2), may be combined into $[e]^{\approx} \cap {\sim}C = \emptyset$. This follows from Lemma A.2 and that $[e]^{\approx} = [e]^{\curlyeqprec} \cup [e]^{\curlyeqsucc}$. Cond. 2 also implies $[e]^{\approx} \cap C \neq \emptyset$. Thus, the first condition in $\underline{P}_C$ may be removed.

Lemma 7.12. $[e]^{\curlyeqsucc} \cap {\sim}C = \emptyset$ *implies* $[e]^{\approx} \cap C \neq \emptyset$ ■

Proof.
$$\begin{aligned} [e]^{\curlyeqsucc} \cap {\sim}C = \emptyset &\Leftrightarrow [e]^{\curlyeqprec} \cup {\sim}({\sim}C) = \emptyset && \text{(Lemma 7.9)} \\ &\Rightarrow [e]^{\approx} \cap {\sim}({\sim}C) \neq \emptyset && \text{(Lemma 7.7b)} \\ &\Leftrightarrow [e]^{\approx} \cap C \neq \emptyset && \text{(Lemma 7.7a)} \end{aligned}$$
□

This results in the following boundary sets.

Definition 7.16 (Boundary sets). Given a set of classes $C \subseteq V_d$, the following sets constitute the upper and the lower approximations of the known and the potential objects of C:

- $\underline{K}_C = \{x \in U \mid [d(x)]^{\approx} \cap {\sim}C = \emptyset\}$
- $\overline{K}_C = \{x \in U \mid [d(x)]^{\curlyeqprec} \cap {\sim}C = \emptyset\}$
- $\underline{P}_C = \{x \in U \mid [d(x)]^{\curlyeqsucc} \cap {\sim}C = \emptyset\}$
- $\overline{P}_C = \{x \in U \mid [d(x)]^{\approx} \cap C \neq \emptyset\}$ ■

7.6.1 Properties of the boundaries

In this section, we will examine the properties of the boundary sets. The first theorem shows that the boundary sets are complementary.

Theorem 7.2. *Given a set of classes $C \subseteq V_d$, the following complements hold:*

a) $\underline{K}_{\sim C} = U - \overline{P}_C$ c) $\underline{P}_{\sim C} = U - \overline{K}_C$

b) $\overline{K}_{\sim C} = U - \underline{P}_C$ d) $\overline{P}_{\sim C} = U - \underline{K}_C$ ■

Proof.

a)
$$\begin{aligned}\underline{K}_{\sim C} &= \{x \in U \mid [d(x)]^{\approx} \cap \sim(\sim C) = \emptyset\}\\ &= U - \{x \in U \mid [d(x)]^{\approx} \cap \sim(\sim C) \neq \emptyset\}\\ &= U - \{x \in U \mid [d(x)]^{\approx} \cap C \neq \emptyset\} && \text{(Lemma 7.7a)}\\ &= U - \overline{P}_C\end{aligned}$$

b)
$$\begin{aligned}\overline{K}_{\sim C} &= \{x \in U \mid [d(x)]^{\preccurlyeq} \cap \sim(\sim C) = \emptyset\} = U - \{x \in U \mid [d(x)]^{\preccurlyeq} \cap \sim(\sim C) \neq \emptyset\}\\ &= U - \{x \in U \mid [d(x)]^{\succcurlyeq} \cap \sim C = \emptyset\} && \text{(Lemma 7.9)}\\ &= U - \underline{P}_C\end{aligned}$$

c)
$$\begin{aligned}\underline{P}_{\sim C} &= \{x \in U \mid [d(x)]^{\succcurlyeq} \cap \sim(\sim C) = \emptyset\} = U - \{x \in U \mid [d(x)]^{\succcurlyeq} \cap \sim(\sim C) \neq \emptyset\}\\ &= U - \{x \in U \mid [d(x)]^{\preccurlyeq} \cap \sim C = \emptyset\} && \text{(Lemma 7.10)}\\ &= U - \overline{K}_C\end{aligned}$$

d) $\overline{P}_{\sim C} = \{x \in U \mid [d(x)]^{\approx} \cap \sim C \neq \emptyset\} = U - \{x \in U \mid [d(x)]^{\approx} \cap \sim C = \emptyset\} = U - \underline{K}_C$

□

It follows from this theorem that the boundary set on $\sim\sim C$ is equal to the boundary set on C. So, applying double negation to C produces the same known and potential sets as C.

Corollary 7.1.

a) $\underline{K}_{\sim(\sim C)} = \underline{K}_C$ c) $\underline{P}_{\sim(\sim C)} = \underline{P}_C$

b) $\overline{K}_{\sim(\sim C)} = \overline{K}_C$ d) $\overline{P}_{\sim(\sim C)} = \overline{P}_C$ ■

Proof. Using Theorem 7.2b and 7.2c, we have for option b that $\overline{K}_{\sim(\sim C)} = U - \underline{P}_{\sim C} = U - (U - \overline{K}_C) = \overline{K}_C$. The other propositions are proven in the same manner by applying Theorem 7.2. □

We will now examine the properties of the boundaries with respect to unions and intersections. We begin with unions. Given the discussion in Section 7.6, it should not be a surprise that the boundary sets, except for $\overline{P}_C$, do not maintain unions. However, we may prove a weaker property. For each boundary set, it holds that the boundary set on $C \cup D$ is a superset of the union of the individual boundary sets on C and D (where C and D are sets of classes).

Lemma 7.13. *Given two set of classes C and D, the following properties hold:*

a) $\underline{K}_{C\cup D} \supseteq \underline{K}_C \cup \underline{K}_D$

b) $\overline{K}_{C\cup D} \supseteq \overline{K}_C \cup \overline{K}_D$

c) $\underline{P}_{C\cup D} \supseteq \underline{P}_C \cup \underline{P}_D$

d) $\overline{P}_{C\cup D} = \overline{P}_C \cup \overline{P}_D$ ■

Proof.

a)
$$\begin{aligned}\underline{K}_{C\cup D} &= \{x \in U \mid [d(x)]^{\approx} \cap {\sim}(C \cup D) = \emptyset\} \\ &= \{x \in U \mid [d(x)]^{\approx} \cap ({\sim}C \cap {\sim}D) = \emptyset\} && \text{(Lemma 7.2)} \\ &\supseteq \{x \in U \mid [d(x)]^{\approx} \cap {\sim}C = \emptyset \text{ or } [d(x)]^{\approx} \cap {\sim}D = \emptyset\} && \text{(Lemma A.3)} \\ &= \underline{K}_C \cup \underline{K}_D\end{aligned}$$

b)
$$\begin{aligned}\overline{K}_{C\cup D} &= \{x \in U \mid [d(x)]^{\preccurlyeq} \cap {\sim}(C \cup D) = \emptyset\} \\ &= \{x \in U \mid [d(x)]^{\preccurlyeq} \cap ({\sim}C \cap {\sim}D) = \emptyset\} && \text{(Lemma 7.2)} \\ &\supseteq \{x \in U \mid [d(x)]^{\preccurlyeq} \cap {\sim}C = \emptyset \text{ or } [d(x)]^{\preccurlyeq} \cap {\sim}D = \emptyset\} && \text{(Lemma A.3)} \\ &= \overline{K}_C \cup \overline{K}_D\end{aligned}$$

c)
$$\begin{aligned}\underline{P}_{C\cup D} &= \{x \in U \mid [d(x)]^{\succcurlyeq} \cap {\sim}(C \cup D) = \emptyset\} \\ &= \{x \in U \mid [d(x)]^{\succcurlyeq} \cap ({\sim}C \cap {\sim}D) = \emptyset\} && \text{(Lemma 7.2)} \\ &\supseteq \{x \in U \mid [d(x)]^{\succcurlyeq} \cap {\sim}C = \emptyset \text{ or } [d(x)]^{\succcurlyeq} \cap {\sim}D = \emptyset\} && \text{(Lemma A.3)} \\ &= \underline{P}_C \cup \underline{P}_D\end{aligned}$$

d)
$$\begin{aligned}\overline{P}_{C\cup D} &= \{x \in U \mid [d(x)]^{\approx} \cap (C \cup D) \neq \emptyset\} \\ &= \{x \in U \mid [d(x)]^{\approx} \cap C \neq \emptyset \text{ or } [d(x)]^{\approx} \cap D \neq \emptyset\} && \text{(Lemma A.2)} \\ &= \overline{P}_C \cup \overline{P}_D\end{aligned}$$
□

None of the boundary sets maintains intersections. Still, we can show that a boundary set on $C \cap D$ is a subset of the intersection of the boundary sets on C and D.

Lemma 7.14. *Given two set of classes C and D, the following properties hold:*

a) $\underline{K}_{C\cap D} \subseteq \underline{K}_C \cap \underline{K}_D$

b) $\overline{K}_{C\cap D} \subseteq \overline{K}_C \cap \overline{K}_D$

c) $\underline{P}_{C\cap D} \subseteq \underline{P}_C \cap \underline{P}_D$

d) $\overline{P}_{C\cap D} \subseteq \overline{P}_C \cap \overline{P}_D$ ■

Proof.

a) This follows directly from options b and c as $\underline{K}_C$ is an intersection of $\overline{K}_C$ and $\underline{P}_C$.

b) $$\begin{aligned}\overline{K}_{C\cap D} &= \{x \in U \mid [d(x)]^{\preccurlyeq} \cap {\sim}(C \cap D) = \emptyset\} \\ &\subseteq \{x \in U \mid [d(x)]^{\preccurlyeq} \cap ({\sim}C \cup {\sim}D) = \emptyset\} && \text{(Lemmas 7.3 and A.1)} \\ &= \{x \in U \mid [d(x)]^{\preccurlyeq} \cap {\sim}C = \emptyset \text{ and } [d(x)]^{\preccurlyeq} \cap {\sim}D = \emptyset\} && \text{(Lemma A.2)} \\ &= \overline{K}_C \cap \overline{K}_D\end{aligned}$$

c) $$\begin{aligned}\underline{P}_{C\cap D} &= \{x \in U \mid [d(x)]^{\succcurlyeq} \cap {\sim}(C \cap D) = \emptyset\} \\ &\subseteq \{x \in U \mid [d(x)]^{\succcurlyeq} \cap ({\sim}C \cup {\sim}D) = \emptyset\} && \text{(Lemmas 7.3 and A.1)} \\ &= \{x \in U \mid [d(x)]^{\succcurlyeq} \cap {\sim}C = \emptyset \text{ and } [d(x)]^{\succcurlyeq} \cap {\sim}D = \emptyset\} && \text{(Lemma A.2)} \\ &= \underline{P}_C \cap \underline{P}_D\end{aligned}$$

d) $$\begin{aligned}\overline{P}_{C\cap D} &= \{x \in U \mid [d(x)]^{\approx} \cap (C \cap D) \neq \emptyset\} \\ &\subseteq \{x \in U \mid [d(x)]^{\approx} \cap C \neq \emptyset \text{ and } [d(x)]^{\approx} \cap D \neq \emptyset\} && \text{(Lemma A.3)} \\ &= \overline{P}_C \cap \overline{P}_D\end{aligned}$$ □

Note that the properties in Lemma 7.13 and Lemma 7.14 are similar to the properties of the upper and the lower approximations, which were given in Section 4.3. The lower approximation does not maintain unions, and the upper approximation does not maintain intersections. Similar results hold also for rough set memberships functions where the membership functions on $X \cup Y$ and $X \cap Y$ cannot be computed from the memberships functions on X and Y (see [136] for more details).

With regard to intersections it should be mentioned that $\overline{K}_{C\cap D}$ and $\overline{K}_C \cap \overline{K}_D$ describe different concepts (the situation is similar for the other boundary sets). $C \cap D$ is the set of *classes* that C and D have in common, and $\overline{K}_{C\cap D}$ is the set of objects that belong to these classes. $\overline{K}_C \cap \overline{K}_D$, on the other hand, is the set of *objects* that C and D have in common. For example, if we consider the DAG in Figure 7.1 and set $C = \{e_1\}$ and $D = \{d_2\}$, then $\overline{K}_{C\cap D}$ will be empty, while $\overline{K}_C \cap \overline{K}_D$ will contain the objects that are labeled to e_1. In this case, $\overline{K}_C \cap \overline{K}_D$ seems more useful than $\overline{K}_{C\cap D}$ and appeals more to our intuition. In particular, $\overline{K}_{C\cap D}$ ignores the relationships between the classes. It is therefore a question if intersection is an interesting operator for classes. An operator that takes the relationships into account would probably be more useful. For example, we could define an operator like $C \Cap D = \{c \in C \cup D \mid c \approx e$, for some $e \in C$ and $c \approx f$, for some $f \in D\}$. However, we will not need such an operator in this thesis and will not pursue this issue any further.

7.6.2 Some special cases

When the DAG is a tree

A tree is a special case of a DAG. Many ontologies for the characterization of genes are represented as taxonomies, and most of these are trees. So, if we choose another ontology than the Gene Ontology, it might be only a tree. We therefore investigate what restrictions that may be made to the framework if the ontology is a tree.

A rooted DAG is a tree if there is only one path from any class e in the DAG to the root class. This means that all classes in the above set $[e]^{\succcurlyeq}$ must be related.

Definition 7.17 (Tree). The DAG is a *tree* if it is rooted with a root class $\top$ (see Definition A.16) and for each class $e \in V_d - \{\top\}$ there is a *unique* path $e = e_1 \prec e_2 \prec \cdots \prec e_n = \top$ from e to $\top$. ■

Lemma 7.15. *If the DAG is a tree, then for each class $e \in V_d$ it holds that all superclasses of e are related: For all $x, y \in [e]^{\succcurlyeq} \Rightarrow x \approx y$* ■

Proof. Assume that the lemma is false. Then x', y' are in $[e']^{\succcurlyeq}$ for some $e' \in V_d$, and $x' \not\approx y'$. Since the DAG is a tree, there is a unique path $e' = e_1 \prec e_2 \prec \cdots \prec e_n = \top$. We may assume without loss of generality that x' is on this path. This means that y' is not on the path since $x' \not\approx y'$. However, $y' \succcurlyeq e'$ since $y \in [e']^{\succcurlyeq}$ and $\top \succcurlyeq y'$ since the DAG is rooted (see Definition A.16). There must consequently be a path from e' to $\top$ that passes through y'. Hence, we have two paths from e' to $\top$ and obtain a contradiction since the path between e' and $\top$ was supposed to be unique. □

A tree has some special properties. In particular, a class has only one immediate superclass. This means that there cannot be any conflict where a class has one superclass that belongs to C and another superclass that belongs to the complement $\sim C$. In other words, no DAG-inconsistency may occur. The difference between the upper and the lower approximations of K_C and P_C vanishes.

Lemma 7.16. *If the DAG forms a tree, then for any $e \in V_d$,*

a) $[e]^{\approx} \cap C \neq \emptyset$ *implies* $[e]^{\succcurlyeq} \cap \sim C = \emptyset$

b) $[e]^{\preccurlyeq} \cap \sim C = \emptyset$ *implies* $[e]^{\succcurlyeq} \cap \sim C = \emptyset$ ■

Proof.

a) If $[e]^{\approx} \cap C \neq \emptyset$, then there is some $c' \in C$ such that $e \approx c'$. If $e \succcurlyeq c'$, every $x \in [e]^{\succcurlyeq}$ is related to c' since $x \succcurlyeq e \succcurlyeq c'$. If $c' \succcurlyeq e$, then $c' \in [e]^{\succcurlyeq}$ and Lemma 7.15 implies that any $x \in [e]^{\succcurlyeq}$ is related to c'. Thus, no $x \in [e]^{\succcurlyeq}$ is in $\sim C$, and $[e]^{\succcurlyeq} \cap \sim C = \emptyset$.

b)
$$\begin{aligned}
[e]^{\preccurlyeq} \cap \sim C = \emptyset &\Leftrightarrow [e]^{\succcurlyeq} \cap \sim(\sim C) \neq \emptyset && \text{(Lemma 7.10)}\\
&\Rightarrow [e]^{\approx} \cap \sim(\sim C) \neq \emptyset && \text{(Lemma 7.7c)}\\
&\Leftrightarrow [e]^{\approx} \cap C \neq \emptyset && \text{(Lemma 7.7a)}\\
&\Rightarrow [e]^{\succcurlyeq} \cap \sim C = \emptyset && \text{(From option a)}
\end{aligned}$$

□

Theorem 7.3. *If the DAG forms a tree, $\overline{K}_C = \underline{K}_C$ and $\overline{P}_C = \underline{P}_C$.* ■

Proof. $\overline{K}_C = \underline{K}_C$ follows immediately from Lemma 7.16b. $\overline{P}_C = \underline{P}_C$ follows from Lemma 7.16a (and Lemma 7.12). □

When the DAG is well-defined

In Section 7.6, we had to make several adjustments to the boundary sets since they were not complementary. In particular, it was necessary to assign the objects in a class e to the known set of C if all of its subclasses were related to a class in C (even though e was not a subclass of any class in C). Similarly, if a class c was a lonely child of a class f, f had to be assigned to the known set of c.

It is possible, however, to avoid these complications, if we consider only single classes and assume that the DAG is well-defined such that there are no lonely children. In this case, the boundary sets may be simplified. We will consider these simplifications in this section. We begin with a definition of what it means for a DAG to be well-defined.

Definition 7.18 (Well-defined DAG). A DAG is well-defined if it holds for every non-leaf class e that for each subclass of e there is another subclass of e such that these subclasses are unrelated. In other words, for any $e, x \in V_d$ if $e \succ x$ then there is some $y \in V_d$ such that $e \succ y$ and $x \not\approx y$. ■

We may now define the alternative boundary sets for a single class.

Definition 7.19 (Boundary sets for a single class). Given a (single) class c, the following sets constitute the upper and the lower approximations of the known and the potential objects of c:

- $\underline{K}^*_c = \{x \in U \mid c \succsim d(x) \text{ and } \neg\exists f \in {\sim}\{c\}\ f \succsim d(x)\}$
  $= \{x \in U \mid [d(x)]^{\succsim} \cap \{c\} \neq \emptyset \text{ and } [d(x)]^{\succsim} \cap {\sim}\{c\} = \emptyset\}$
- $\overline{K}^*_c = \{x \in U \mid c \succsim d(x)\} = \{x \in U \mid [d(x)]^{\succsim} \cap \{c\} \neq \emptyset\}$
- $\underline{P}^*_c = \{x \in U \mid \neg\exists f \in {\sim}\{c\}\ f \succsim d(x)\} = \{x \in U \mid [d(x)]^{\succsim} \cap {\sim}\{c\} = \emptyset\}$
- $\overline{P}^*_c = \{x \in U \mid c \approx d(x)\} = \{x \in U \mid [d(x)]^{\approx} \cap \{c\} \neq \emptyset\}$ ■

It follows immediately from this definition and Definition 7.16 that $\underline{P}^*_c = \underline{P}_{\{c\}}$ and $\overline{P}^*_c = \overline{P}_{\{c\}}$. So, the single-class potential sets may be used even when the DAG is not well-defined. The problem lies with the known sets where only $\underline{K}^*_c \subseteq \underline{K}_{\{c\}}$ and $\overline{K}^*_c \subseteq \overline{K}_{\{c\}}$ hold generally. Still, if the DAG is well-defined, we may prove the following lemmas. They imply that the original set-based known sets and the alternative single-class known sets must be equal for a single class.

Lemma 7.17. *If the DAG is well-defined, then* $[e]^{\precsim} \cap {\sim}\{c\} = \emptyset \Leftrightarrow [e]^{\succsim} \cap \{c\} \neq \emptyset$. ■

Proof.

$\Leftarrow$: Always true, since $[e]^{\succsim} \cap {\sim}({\sim}\{c\}) \neq \emptyset$ follows from $[e]^{\succsim} \cap \{c\} \neq \emptyset$ by Lemma 7.7e, and $[e]^{\precsim} \cap {\sim}\{c\} = \emptyset$ follows from $[e]^{\succsim} \cap {\sim}({\sim}\{c\}) \neq \emptyset$ by Lemma 7.10.

⇒: If $[e]^{\preccurlyeq} \cap \sim\{c\} = \emptyset$, then $e \notin \sim\{c\}$ such that $e \approx c$. If $e \succ c$, there is also some $e \succ y$ such that $y \not\approx c$ since the DAG is well-defined. Then $y \in \sim\{c\}$ and $[e]^{\preccurlyeq} \cap \sim\{c\}$ is not empty since $y \in [e]^{\preccurlyeq}$. Thus, $c \succcurlyeq e$ must hold, and this is equivalent to $[e]^{\succcurlyeq} \cap \{c\} \neq \emptyset$. □

Theorem 7.4. *If the DAG is well-defined, then*

a) $\underline{K}^*_c = \underline{K}_{\{c\}}$

b) $\overline{K}^*_c = \overline{K}_{\{c\}}$ ■

Proof.

a) $[d(x)]^{\approx} \cap \sim\{c\} = \emptyset$ (in $\underline{K}_{\{c\}}$) is equivalent to $[d(x)]^{\preccurlyeq} \cap \sim\{c\} = \emptyset$ and $[d(x)]^{\succcurlyeq} \cap \sim\{c\} = \emptyset$. Lemma 7.17 states that $[d(x)]^{\preccurlyeq} \cap \sim\{c\} = \emptyset \Leftrightarrow [d(x)]^{\succcurlyeq} \cap \{c\} \neq \emptyset$. $[d(x)]^{\succcurlyeq} \cap \sim\{c\} = \emptyset$ is obviously true in both sets.

b) Follows immediately from Lemma 7.17. □

The complement of a single class is a set of classes. Thus, we need boundary sets that apply to a set of classes in order to find the known and the potential objects of the complement. Such boundary set could perhaps be found by taking the union of the single-class boundary sets, which belong to the classes in the complement. However, the previous discussion and Lemma 7.13 entails that a union of single-class boundary sets are only a subset of the corresponding set-based boundary set. This suggests that the complement of a single class may not be found with the single-class boundary sets. So their utility may appear limited.

However, the complement is a very special set so that the single-class boundary sets may still be useful. When a set-based boundary set is applied to a set of classes, it may add some classes that would not be included by the union of the single-class boundary set (This is the reason why a set-based boundary set is superset of the union of the corresponding single-class boundary sets). However, no classes will be added if a boundary set is applied to the complement since the complement has a special property; any class that possibly could be added by the boundary set is already in the complement. The union of single-class boundary sets is thus equal to the set-based boundary set in this case. This will be proven in Theorem 7.5.

This definition defines the boundary sets for the complement by using the union of single-class boundary sets.

Definition 7.20 (Boundary set for complement). Let C be a set of classes. The following sets denote the the upper and the lower approximations of the known and the potential objects of $\sim C$:

- $\underline{K}^*_{\sim C} = \bigcup_{c \in \sim C} \underline{K}^*_c$
- $\overline{K}^*_{\sim C} = \bigcup_{c \in \sim C} \overline{K}^*_c$

- $\underline{P}^*_{\sim C} = \bigcup_{c \in \sim C} \underline{P}^*_c$
- $\overline{P}^*_{\sim C} = \bigcup_{c \in \sim C} \overline{P}^*_c$ ■

The following lemmas prove several properties that we will need in Theorem 7.5.

Lemma 7.18. *$[e]^{\succcurlyeq} \cap \{c\} \neq \emptyset$ holds for some $c \in \sim C$ iff $[e]^{\preccurlyeq} \cap \sim(\sim C) = \emptyset$.* ■

Proof. The statement that $[e]^{\succcurlyeq} \cap \{c\} \neq \emptyset$ holds for some $c \in \sim C$ is equivalent to $[e]^{\succcurlyeq} \cap \left(\bigcup_{c \in \sim C} \{c\}\right) \neq \emptyset$ by Lemma A.2, and $\sim C = \bigcup_{c \in \sim C} \{c\}$. $[e]^{\succcurlyeq} \cap \sim C \neq \emptyset$ is again equivalent to $[e]^{\preccurlyeq} \cap \sim(\sim C) = \emptyset$ by Lemma 7.9. □

Lemma 7.19. *If $[e]^{\succcurlyeq} \cap \sim\{c\} = \emptyset$ holds for some $c \in C$ then $[e]^{\succcurlyeq} \cap \sim C = \emptyset$.* ■

Proof. The statement that $[e]^{\succcurlyeq} \cap \sim\{c\} = \emptyset$ holds for some $c \in C$ implies $[e]^{\succcurlyeq} \cap \bigcap_{c \in C} \sim\{c\} = \emptyset$ by Lemma A.3. Lemma 7.1 states that $\sim C = \bigcap_{c \in C} \sim\{c\}$. Hence, $[e]^{\succcurlyeq} \cap \sim C = \emptyset$ is equivalent to $[e]^{\succcurlyeq} \cap \bigcap_{c \in C} \sim\{c\} = \emptyset$. □

Lemma 7.20. *If $[e]^{\succcurlyeq} \cap \sim(\sim C) = \emptyset$ then $[e]^{\succcurlyeq} \cap \sim\{c\} = \emptyset$ holds for some $c \in \sim C$.* ■

Proof. If $[e]^{\succcurlyeq} \cap \sim(\sim C) = \emptyset$ holds, then every $x \in [e]^{\succcurlyeq}$ must be related to some $y \in \sim C$ such that $x \notin \sim(\sim C)$. In particular, e must be related to some $y' \in \sim C$. Then, $e \succcurlyeq y'$ or $y' \succcurlyeq e$ holds.

1. If $e \succcurlyeq y'$, all $x \in [e]^{\succcurlyeq}$ must be above or equal to y' since $x \succcurlyeq e \succcurlyeq y'$. Then $x \approx y'$ such that no x is in $\sim\{y'\}$. Hence, $[e]^{\succcurlyeq} \cap \sim\{y'\} = \emptyset$.
2. If $y' \succcurlyeq e$, we may prove that e is also in $\sim C$. Assume that this is false such that $e \notin \sim C$. Then some $c \in C$ must be related to e. So $e \succcurlyeq c$ or $c \succcurlyeq e$ must hold.
   (a) If $e \succcurlyeq c$, then $y' \succcurlyeq e \succcurlyeq c$ and $y' \approx c$. So y' cannot be in $\sim C$. Contradiction.
   (b) If $c \succcurlyeq e$, then $c \in [e]^{\succcurlyeq}$, and $[e]^{\succcurlyeq} \cap \sim(\sim C)$ cannot be not empty since $C \subseteq \sim(\sim C)$. Contradiction.

   Hence, $e \in \sim C$, and it follows that $[e]^{\succcurlyeq} \cap \sim\{e\} = \emptyset$ since no $x \in [e]^{\succcurlyeq}$ is in $\sim\{e\}$. □

Lemma 7.21. *If $[e]^{\succcurlyeq} \cap \{c_1\} \neq \emptyset$ holds for some $c_1 \in \sim C$ and $[e]^{\succcurlyeq} \cap \sim\{c_2\} = \emptyset$ holds for some $c_2 \in \sim C$ then $e \in \sim C$.* ■

Proof. Assume false. Then $e \notin \sim C$ and $[e]^{\succcurlyeq} \cap \{c_1'\} \neq \emptyset$ and $[e]^{\succcurlyeq} \cap \sim\{c_2'\} = \emptyset$ holds for some $c_1', c_2' \in \sim C$. Then e must be related to some $y \in C$ (since $e \notin \sim C$). Then $e \succcurlyeq y$ or $y \succcurlyeq e$ must hold.

1. If $e \succcurlyeq y$, then $c_1' \succcurlyeq e \succcurlyeq y$ since $[e]^{\succcurlyeq} \cap \{c_1'\} \neq \emptyset$. So $c_1' \approx y$, and c_1' cannot be in $\sim C$. Contradiction.
2. If $y \succcurlyeq e$, then $y \in [e]^{\succcurlyeq}$. From $[e]^{\succcurlyeq} \cap \sim\{c_2'\} = \emptyset$, we have that y cannot be in $\sim\{c_2'\}$. Hence, $y \approx c_2'$, and c_2' cannot be in $\sim C$. Contradiction. □

Theorem 7.5.

a) $\underline{K}^*_{\sim C} = \underline{K}_{\sim C}$

b) $\overline{K}^*_{\sim C} = \overline{K}_{\sim C}$

c) $\underline{P}^*_{\sim C} = \underline{P}_{\sim C}$

d) $\overline{P}^*_{\sim C} = \overline{P}_{\sim C}$ ■

Proof.

a) We have that

$$\begin{aligned}\underline{K}^*_{\sim C} &= \bigcup_{c \in \sim C} \underline{K}^*_c \\ &= \{x \in U \mid [d(x)]^{\succeq} \cap \{c\} \neq \emptyset \text{ and } [d(x)]^{\succcurlyeq} \cap \sim\{c\} = \emptyset, \text{ for some } c \in \sim C\}\end{aligned}$$

The condition

$$[d(x)]^{\succeq} \cap \{c\} \neq \emptyset \text{ and } [d(x)]^{\succcurlyeq} \cap \sim\{c\} = \emptyset, \text{ for some } c \in \sim C \qquad \text{(Cond. 3)}$$

is equivalent to

$$\begin{gathered}[d(x)]^{\succeq} \cap \{c_1\} \neq \emptyset, \text{for some } c_1 \in \sim C \text{ and} \\ [d(x)]^{\succcurlyeq} \cap \sim\{c_2\} = \emptyset, \text{for some } c_2 \in \sim C\end{gathered} \qquad \text{(Cond. 4)}$$

Condition 3 implies condition 4 (Just set $c_1 = c_2 = c$). Lemma 7.21 says that if condition 4 is true, then $d(x) \in \sim C$. Since $[d(x)]^{\succeq} \cap \{d(x)\} \neq \emptyset$ and $[d(x)]^{\succcurlyeq} \cap \sim\{d(x)\} = \emptyset$ are always true, the condition 3 must be true. Hence,

$$\underline{K}^*_{\sim C} = \left\{ x \in U \;\middle|\; \begin{array}{l} [d(x)]^{\succeq} \cap \{c_1\} \neq \emptyset, \text{ for some } c_1 \in \sim C \text{ and} \\ [d(x)]^{\succcurlyeq} \cap \sim\{c_2\} = \emptyset, \text{ for some } c_2 \in \sim C \end{array} \right\}$$

The first part of the condition in this set is equivalent to $[d(x)]^{\preccurlyeq} \cap \sim(\sim C) = \emptyset$ by Lemma 7.18, and the second part is equivalent to $[d(x)]^{\succcurlyeq} \cap \sim(\sim C) = \emptyset$ by Lemma 7.19 (substitute $\sim C$ for C) and Lemma 7.20. So,

$$\begin{aligned}\underline{K}^*_{\sim C} &= \{x \in U \mid [d(x)]^{\preccurlyeq} \cap \sim(\sim C) = \emptyset \text{ and } [d(x)]^{\succcurlyeq} \cap \sim(\sim C) = \emptyset\} \\ &= \{x \in U \mid [d(x)]^{\approx} \cap \sim(\sim C) = \emptyset\} = \underline{K}_{\sim C}\end{aligned}$$

b) $\overline{K}^*_{\sim C} = \bigcup_{c \in \sim C} \overline{K}^*_c = \{x \in U \mid [d(x)]^{\succeq} \cap \{c\} \neq \emptyset, \text{ for some } c \in \sim C\}$
$= \{x \in U \mid [d(x)]^{\preccurlyeq} \cap \sim(\sim C) = \emptyset\} = \overline{K}_{\sim C}$

Follows from Lemma 7.18.

c) $\underline{P}^*_{\sim C} = \bigcup_{c \in \sim C} \underline{P}^*_c = \{x \in U \mid [d(x)]^{\succcurlyeq} \cap \sim\{c\} = \emptyset, \text{ for some } c \in \sim C\}$
$= \{x \in U \mid [d(x)]^{\succcurlyeq} \cap \sim(\sim C) = \emptyset\} = \underline{P}_{\sim C}$

Follows from Lemma 7.19 (insert $\sim C$ for C) and 7.20

d) Follows immediately from the definitions (and Lemma 7.13d). □

Thus, if one assumes that the DAG is well-defined and considers only single classes and their complements, one may actually use the alternative boundary sets instead of the set-based boundaries. We will use these boundary sets in Chapter 8.

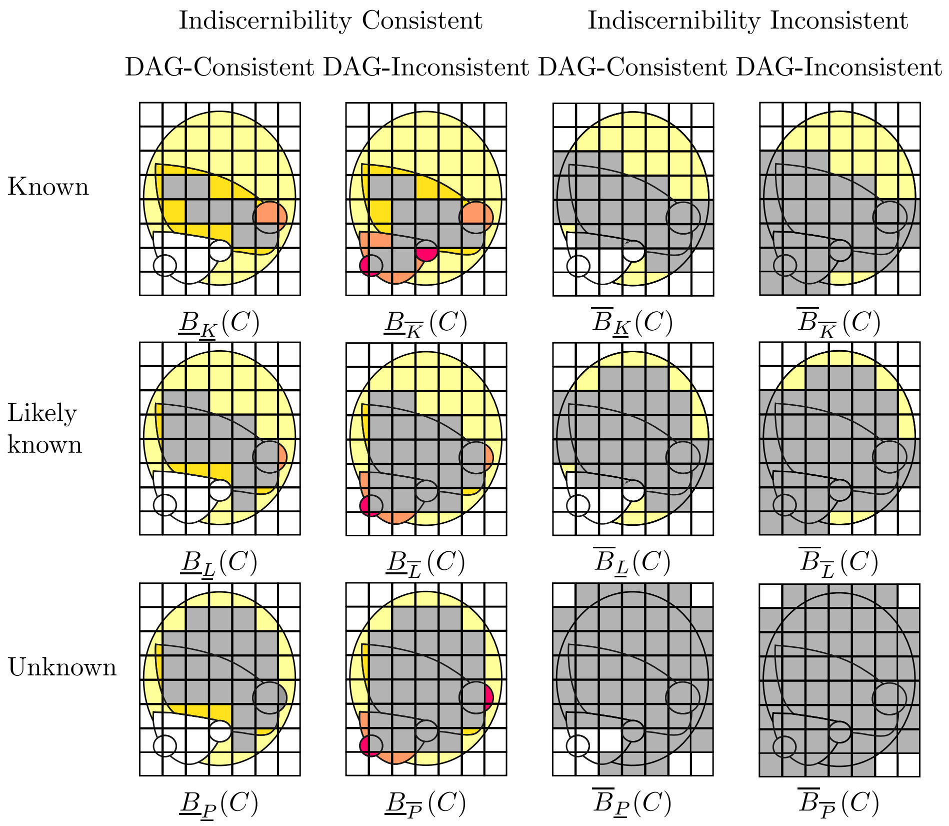

Indiscernibility Consistent
Indiscernibility Inconsistent
DAG-Consistent
DAG-Inconsistent
DAG-Consistent
DAG-Inconsistent
Known
$\underline{B}_{\underline{K}}(C)$
$\underline{B}_{\overline{K}}(C)$
$\overline{B}_{\underline{K}}(C)$
$\overline{B}_{\overline{K}}(C)$
Likely known
$\underline{B}_{\underline{L}}(C)$
$\underline{B}_{\overline{L}}(C)$
$\overline{B}_{\underline{L}}(C)$
$\overline{B}_{\overline{L}}(C)$
Unknown
$\underline{B}_{\underline{P}}(C)$
$\underline{B}_{\overline{P}}(C)$
$\overline{B}_{\underline{P}}(C)$
$\overline{B}_{\overline{P}}(C)$

Figure 7.2: *An illustration of the operators on the DAG in Figure 7.1 where* $C = \{c\}$. The unknown region is displayed as yellow, the regions corresponding to c and its subclasses are shown as orange or red, and the negative region is white. The region covered by each operator is dark gray.

7.7 Set approximations for a DAG-decision system

The boundary sets $\underline{K}_C$, $\overline{K}_C$, $\underline{P}_C$, and $\overline{P}_C$ solve only the part of the problem that is related to the DAG. They do not consider the uncertainty (due to noise), which is handled by the standard rough set approximations. The sets may be inconsistent if they are considered in terms of the elementary sets that are created by an indiscernibility relation. In order to take this kind of inconsistency into account, we use the standard rough set approximations on the boundary sets:

- $\underline{B}_X(C) = \underline{B}\,X_C = \{x \in U \mid [x]_B \subseteq X_C\}$
- $\overline{B}_X(C) = \overline{B}\,X_C = \{x \in U \mid [x]_B \cap X_C \neq \emptyset\}$

We create two operators for each boundary set by replacing X with $\underline{K}$, $\overline{K}$, $\underline{P}$, or $\overline{P}$.

The operators are illustrated in Figure 7.2. However, they do not capture our intuition completely when *IND*(B) is an indiscernibility relation on the conditional attributes. The lower approximations $\underline{B}_{\underline{K}}(C)$ and $\underline{B}_{\overline{K}}(C)$ are very conservative, and it seems that more elementary sets should belong to C. In particular, the elementary sets where at least some of the objects are known (with respect to C) and the remaining ones are unknown, are not included in $\underline{B}_{\underline{K}}(C)$ and $\underline{B}_{\overline{K}}(C)$. This is unfortunate since the objects in these elementary sets probably belong to C. Some of the objects are already known to belong C. So, if we assume that the unknown objects in these sets belong to the complement classes, we will introduce new inconsistencies into the decision system. If all objects in these sets belong to C, on the other hand, no such inconsistencies will occur. Hence, the unknown objects in these elementary sets most likely belong to C since this leads to the fewest number of inconsistencies. We call these elementary sets for *C-likely-known* sets.

A key feature of the *C-likely-known* sets is that they are consistent with the potential sets, but inconsistent with the known sets. Thus, the upper approximations of the known sets and the lower approximations of the potential sets contain the *C-likely-known* sets. However, these approximations contain also other objects that should not be in a lower approximation of the known sets. $\overline{B}_{\underline{K}}(C)$ and $\overline{B}_{\overline{K}}(C)$ cover the inconsistent elementary sets where some objects are known to belong to $\sim C$. $\underline{B}_{\overline{P}}(C)$ and $\underline{B}_{\underline{P}}(C)$ cover completely unknown elementary sets, i.e., sets consisting of only unknown objects. The intersection of $\overline{B}_{\overline{K}}(C)$ and $\underline{B}_{\overline{P}}(C)$, on the other hand, contains only completely known and *C-likely-known* elementary sets (The situation is similar for $\overline{B}_{\underline{K}}(C)$ and $\underline{B}_{\underline{P}}(C)$). Hence, C-likely-known sets may be added to $\underline{B}_{\underline{K}}(C)$ and $\underline{B}_{\overline{K}}(C)$ by using combinations of these approximations. This motivates us to define the following approximations.

Definition 7.21 (Approximations with C-likely-known sets).

a) $\underline{B}_{\underline{L}}(C) = \underline{B}_{\underline{P}}(C) \cap \overline{B}_{\underline{K}}(C) = \{x \in U \mid [x]_B \subseteq \underline{P}_C \text{ and } [x]_B \cap \underline{K}_C \neq \emptyset\}$

b) $\underline{B}_{\overline{L}}(C) = \underline{B}_{\overline{P}}(C) \cap \overline{B}_{\overline{K}}(C) = \{x \in U \mid [x]_B \subseteq \overline{P}_C \text{ and } [x]_B \cap \overline{K}_C \neq \emptyset\}$

c) $\overline{B}_{\underline{L}}(C) = \underline{B}_{\underline{P}}(C) \cup \overline{B}_{\underline{K}}(C) = \{x \in U \mid [x]_B \subseteq \underline{P}_C \text{ or } [x]_B \cap \underline{K}_C \neq \emptyset\}$

d) $\overline{B}_{\overline{L}}(C) = \underline{B}_{\overline{P}}(C) \cup \overline{B}_{\overline{K}}(C) = \{x \in U \mid [x]_B \subseteq \overline{P}_C \text{ or } [x]_B \cap \overline{K}_C \neq \emptyset\}$ ■

Notice that $\overline{B}_{\underline{L}}(C)$ and $\overline{B}_{\overline{L}}(C)$ contain the same elementary sets as $\overline{B}_{\underline{P}}(C)$ and $\overline{B}_{\overline{P}}(C)$ except for the ($\sim C$)-likely-known sets, which must likely belong to the complement $\sim C$.

7.7.1 Some properties of the approximations

Each approximation has a dual approximation such that the approximation and its dual are complementary.

Definition 7.22 (Dual approximations). Two approximations X and Y are duals denoted as $X \leftrightarrow Y$ if $X(C) = U - Y(\sim C)$ and $X(\sim C) = U - Y(C)$ hold. ■

Corollary 7.2. *The approximations form the following duals.*

a) $\underline{B}_{\underline{K}} \leftrightarrow \overline{B}_{\overline{P}}$ c) $\underline{B}_{\underline{P}} \leftrightarrow \overline{B}_{\overline{K}}$ e) $\underline{B}_{\underline{L}} \leftrightarrow \overline{B}_{\overline{L}}$

b) $\underline{B}_{\overline{K}} \leftrightarrow \overline{B}_{\underline{P}}$ d) $\underline{B}_{\overline{P}} \leftrightarrow \overline{B}_{\underline{K}}$ f) $\underline{B}_{\overline{L}} \leftrightarrow \overline{B}_{\underline{L}}$ ■

Proof. From rough set theory, we have that $\underline{B}(U - X) = U - \overline{B}(X)$ and $\overline{B}(U - X) = U - \underline{B}(X)$ for a set $X \subseteq U$.

a) Follows from these properties and Theorems 7.2a and 7.2d.

b) Follows from these properties and Theorems 7.2b and 7.2c.

c) Follows from these properties and Theorems 7.2b and 7.2c.

d) Follows from these properties and Theorems 7.2a and 7.2d.

e) $\underline{B}_{\underline{L}}(C) = \underline{B}_{\underline{P}}(C) \cap \overline{B}_{\underline{K}}(C)$
$= (U - \overline{B}_{\overline{K}}(\sim C)) \cap (U - \underline{B}_{\overline{P}}(\sim C))$ (from Options c and d)
$= U - (\overline{B}_{\overline{K}}(\sim C) \cup \underline{B}_{\overline{P}}(\sim C)) = U - \overline{B}_{\overline{L}}(\sim C)$

The proof of $\underline{B}_{\underline{L}}(\sim C) = U - \overline{B}_{\overline{L}}(C)$ is similar.

f) $\underline{B}_{\overline{L}}(C) = \underline{B}_{\overline{P}}(C) \cap \overline{B}_{\overline{K}}(C)$
$= (U - \overline{B}_{\underline{K}}(\sim C)) \cap (U - \underline{B}_{\underline{P}}(\sim C))$ (from Options c and d)
$= U - (\overline{B}_{\underline{K}}(\sim C) \cup \underline{B}_{\underline{P}}(\sim C)) = U - \overline{B}_{\underline{L}}(\sim C)$

The proof of $\underline{B}_{\overline{L}}(\sim C) = U - \overline{B}_{\underline{L}}(C)$ is similar. □

The approximations are related such that some approximations are subsets of the other approximations. They may be ordered according to subset inclusion. This order is shown in Figure 7.3 where an arrow from one approximation to another approximation means that the former is a subset of the latter. These properties follow directly from the definitions. So no proof is given.

One may also derive properties for the approximations with regard to unions and intersections of classes. However, these properties are easily proven from the (rough set) properties described in Section 4.3 and the properties of the boundary sets that we established in Section 7.6.1. So, we will not consider them here.

$$\begin{tikzcd}
 & & \overline{B}_{\overline{P}}(C) & \\
\underline{B}_{\overline{P}}(C) \arrow[urr] \arrow[drr] & & & \overline{B}_{\underline{P}}(C) \arrow[ul] \\
 & & \overline{B}_{\overline{L}}(C) \arrow[uu] & \\
 & \underline{B}_{\underline{P}}(C) \arrow[uul] \arrow[drr] \arrow[uurr] & & \\
\underline{B}_{\overline{L}}(C) \arrow[uuu] \arrow[drr] \arrow[uurr] & & & \overline{B}_{\underline{L}}(C) \arrow[uuu] \arrow[uul] \\
 & & \overline{B}_{\overline{K}}(C) \arrow[uuu] & \\
 & \underline{B}_{\underline{L}}(C) \arrow[uul] \arrow[drr] \arrow[uuu] \arrow[uurr] & & \\
\underline{B}_{\overline{K}}(C) \arrow[uuu] \arrow[uurr] & & & \overline{B}_{\underline{K}}(C) \arrow[uul] \arrow[uuu] \\
 & \underline{B}_{\underline{K}}(C) \arrow[ul] \arrow[uu] \arrow[urr] & &
\end{tikzcd}$$

Figure 7.3: *Subset inclusion of the approximations.* An arrow from A to B denotes that $A \subseteq B$.

Algorithms for Learning in a Rooted DAG

8

A previous version of the ensemble method has been published in [115, 113].

8.1 Introduction

The last two chapters discussed the Gene Ontology and introduced a framework for the DAG based on rough set theory. This chapter presents two different algorithms for learning classifiers in a DAG. Both of them create a set of rules in a similar manner to the covering approach in Section 4.6.2. However, the classes and objects are treated differently than in an ordinary covering algorithm. In order to explain these algorithms and motivate the ideas behind them, we need to look more closely into the problem of learning in a DAG.

8.2 What makes learning in a DAG difficult?

It may not be apparent that learning in a DAG is a special problem that cannot be handled by an ordinary learning algorithm. However, the DAG introduces several issues that may get such an algorithm into trouble.

8.2.1 The classes are related

The most obvious problem is the structure between the classes. An ordinary rule learning algorithm assumes that the classes are unrelated. It will try to find rules that discriminate between related classes if it is applied on the DAG. This may result in very specific rules. Specific rules are, however, a problem since they cover few objects and are more likely to be based on artifacts (in the training data) that do not actually contribute to discerning the objects.

For example, some of the objects in two related classes may really be indiscernible, but some of their attribute values may be different due to noise. An ordinary rule learner will, in this case, try to find rules that discern between these objects. In order to do so it will choose the noisy attributes as conditional attributes and may neglect other more relevant attributes. The rules may consequently make poor predictions when applied to a test set.

Hence, an ordinary learning algorithm will be very sensitive to noise and may have a low predictive power. If we do not discriminate between the related classes, on the other hand, we may create more general and better rules.

Example 8.1. In Examples 4.9 and 4.11 we made rules with two RST methods for the for the decision system in Table 4.2. The classes assigned to the genes in this system were actually related, and were taken from the GO process ontology (see Figure 8.1). According to this ontology, `intracellular protein traffic` is a subclass of `cell growth & maintenance` so that the rules need not discriminate between these classes. In this case, we may find a rule like:

$$\langle \texttt{15m-30m,down} \rangle \wedge \langle \texttt{30m-1H,down} \rangle \rightarrow \langle \texttt{Process,cell growth \& maintenance} \rangle$$

This rule covers objects o_3, o_7, o_8, and o_9 and is more general than the corresponding rule found in Examples 4.9 and 4.11 (see Figure 8.2). So it is clearly an improvement on these rules. Still, objects o_3 and o_9 are labeled with subclasses of `cell growth & maintenance`. The predictions made by this rule will be less detailed than the original decision classes of objects o_3 and o_9. So something is lost in this case.

However, we may find more general rules for the subclasses as well. Objects o_8 and o_9 have almost the same information vector, and it is likely that object o_8 belongs to `intracellular protein traffic`. If we do not discern between related classes, we may learn a rule such as:

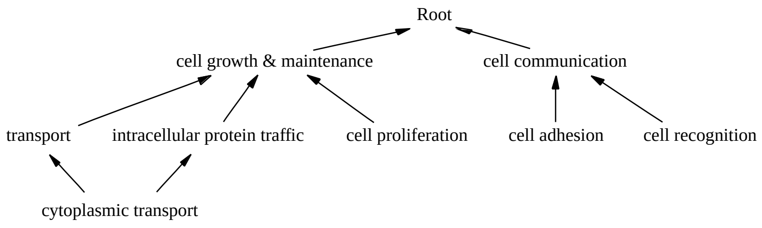
$$\langle \texttt{30m-1H,down} \rangle \wedge \langle \texttt{1H-2H,down} \rangle \rightarrow \langle \texttt{Process,intracellular protein traffic} \rangle$$

This rule covers both objects o_8 and o_9, and it is more general than the corresponding rules in Example 4.9. The rules in this example have the same number of descriptors. However, they try to discern between the objects by means of attribute `0H-15m` for which the objects have a different value. This difference is most likely due to noise such that the rules from Example 4.11 may have a lower prediction power than this rule. Furthermore, the rule gives a more detailed prediction for object o_8. So, we obtain both more accurate and more detailed predictions in this case.

A similar situation occurs for objects o_3 and o_7 where object o_7 is annotated with a superclass of the class of object o_3. Both objects are members of the same elementary set, and we may assume that object o_7 belongs to the same class as object o_3. Hence, the following rule may be found:

$$\langle \texttt{30m-1H,down} \rangle \wedge \langle \texttt{1H-2H,up} \rangle \rightarrow \langle \texttt{Process,intracellular protein traffic} \rangle$$

■



Object	*0H-15m*	*15m-30m*	*30m-1H*	*1H-2H*	*2H-4H*	*Process Name*
o_1	up	up	up	const	const	cell proliferation
o_2	up	up	up	down	down	cell proliferation
o_3	down	down	down	up	up	cell proliferation
o_4	up	up	up	down	down	cell adhesion
o_5	down	up	up	down	up	cell adhesion
o_6	const	const	const	down	down	cell adhesion
o_7	down	down	down	up	up	cell growth & maintenance
o_8	up	down	down	down	down	cell growth & maintenance
o_9	const	down	down	down	down	intracellular protein traffic
o_{10}	down	up	up	up	up	transport
o_{11}	down	down	up	up	up	cytoplasmic transport

Figure 8.1: *A DAG-decision system $\mathcal{A} = \langle U, A, d, \succcurlyeq \rangle$.* The DAG is a small part of the process ontology (rev. 1.221 - 05-Feb-2001). The decision table contains the decision system from Table 4.2

Rules from Example 4.9:

$$\begin{array}{lcll}
\langle\texttt{0H-15m, d}\rangle\wedge\langle\texttt{15m-30m, d}\rangle\wedge\langle\texttt{30m-1H, d}\rangle\wedge\langle\texttt{1H-2H, u}\rangle & \rightarrow & \langle\texttt{Process, cell prolif.}\rangle & \\
 & & \vee\ \langle\texttt{Process, cell growth}\rangle & (o_3, o_7) \\
\langle\texttt{0H-15m, up}\rangle\wedge\langle\texttt{15m-30m, down}\rangle & \rightarrow & \langle\texttt{Process, cell growth}\rangle & (o_8) \\
\langle\texttt{0H-15m, const}\rangle\wedge\langle\texttt{15m-30m, down)}\rangle & \rightarrow & \langle\texttt{Process, intra. p. traffic}\rangle & (o_9)
\end{array}$$

Rules from Example 4.11:

$$\begin{array}{lcll}
\langle\texttt{0H-15m, d}\rangle\wedge\langle\texttt{15m-30m, d}\rangle & \rightarrow & \langle\texttt{Process, cell prolif.}\rangle & (o_3, o_7) \\
\langle\texttt{0H-15m, d}\rangle\wedge\langle\texttt{15m-30m, d}\rangle & \rightarrow & \langle\texttt{Process, cell growth}\rangle & (o_3, o_7) \\
\langle\texttt{0H-15m, u}\rangle\wedge\langle\texttt{15m-30m, d}\rangle\wedge\langle\texttt{30m-1H, d}\rangle\wedge\langle\texttt{1H-2H, d}\rangle & \rightarrow & \langle\texttt{Process, cell growth}\rangle & (o_8) \\
\langle\texttt{0H-15m, c}\rangle\wedge\langle\texttt{15m-30m, d}\rangle\wedge\langle\texttt{30m-1H, d}\rangle\wedge\langle\texttt{1H-2H, d}\rangle & \rightarrow & \langle\texttt{Process, intra. p. traf.}\rangle & (o_9)
\end{array}$$

Figure 8.2: *Rules from Examples 4.9 and 4.11.* The values *up*, *down*, and *const* are represented as *u*, *d*, and *c*, respectively.

8.2.2 The detail level of the annotations varies

Another problem that occurs in the DAG, is the detail level of the annotations. The existing biological knowledge about the processes of genes has a strongly varying detail level. Some annotations refer to leaf classes, but many annotations concern non-leaf classes. This is illustrated in the DAG-decision system in Figure 8.1 where objects o_7 and o_8, for instance, are labeled with the non-leaf class `cell growth & maintenance`.

The variation in the detail level means that we cannot avoid the structure of the ontology such that an ordinary learning algorithm can be applied directly. For example, if all of the annotations referred to leaf classes, we could reduce the problem by only using the leaf classes. However, since an annotation may refer to a non-leaf class, we would lose a lot of annotations if we tried. Moving all non-leaf annotations to the leaf classes is not an alternative, either. Most of the annotations in the leaf classes would belong to non-leaf classes, and these would be shared by many leaf classes. Discerning between these classes would thus be very hard.

Alternatively, we could create a "cut" through the ontology by selecting the most general classes, which contain genes (i.e., `cell growth & maintenance` and `cell adhesion` in Figure 8.1) and move the annotations from the subclasses to these classes. However, the details of the moved annotations would be lost.

Hence, the ontology cannot be reduced to a flat set of unrelated classes without losing whole annotations or annotation details. So the structure of the ontology must be considered during learning and prediction.

8.2.3 There are few objects per class

A rule learning algorithm needs a minimum number of objects in order to learn rules that accurately predict a class. The number of objects in an ordinary training set is therefore much larger than the number of classes. However, the number of genes in a microarray study may be of same magnitude as (or even smaller than) the number of classes in the process ontology. Hence, the number of objects that are available for each class will be very low. It may therefore be impossible to learn accurate rules.

One extreme example is the yeast genome that has about $6,000$ genes. Revision 2.577 of the process ontology, on the other hand, contains about $4,400$ different classes. There will consequently be only 1.34 objects/class. However, the number of genes per class may still be quite small even with a large genome such as the human genome. It contains about $30,000$-$40,000$ genes such that each class would have only 7-9 objects on average.

The number of objects/class may be a little higher in practice. Some of classes in the ontology may not be relevant for a particular organism, and these may be removed. Moreover, a gene may be annotated to more than one class so that several classes may share the same objects. However, there may still be a lot of classes that need to be discerned, and the annotations may be spread throughout the whole ontology so that each class may have very few objects. Finding genuine distinguishing properties in the information vectors may be very hard even if we do *not* discriminate between the related classes.

Thus, it may be necessary to increase the number of objects in each class. This may be achieved by moving the objects from some related classes to a joint class. Rules for the joint class can then learned from the total set of objects[1].

The annotations may be moved either upwards or downwards. Moving objects upwards is the simplest procedure and is always correct. An object x annotated with a class c belongs also to any of the superclasses of c. So, if a superclass of c is predicted for x, a correct prediction has been made even though the prediction is less detailed.

Moving objects downwards is more difficult. Details are gained in this case, but a decision must be made with regard to the subclass to which an annotation should be moved. The situation will not improve if the objects of a class are moved to all of the immediate subclasses. The same objects will just occur in all of the subclasses so that a learning algorithm will be unable to separate between the subclasses by the means of these objects. The discernibility of the subclasses will just remain the same.

So one (or a few) of the subclasses should be selected as target(s) for the move. The only available knowledge for making such a decision is the information vectors. However, objects having a similar information vectors are likely to belong to same class. It seems reasonable to move generally annotated objects to subclasses where the objects have similar information vectors[2]. One way to do this would be to apply a similarity measure and compute the similarity of the information vectors. An object could then be moved to the subclass with the most similar objects according to this measure.

We would like to stress that we are not suggesting that we move all objects to the most general or the most specific classes. This idea was dismissed in Section 8.2.2, and we are not reconsidering it. We are only proposing that the objects are moved to some of the related classes so that the precision and the detail level of the predictions are optimized.

8.2.4 There is a trade-off between detail level and precision

Moving annotations downwards is obviously preferable since more detailed predictions are obtained. However, it may not be possible to get satisfactory precision by moving genes downwards, and better results may be obtained by moving the objects upwards. There is, in fact, a trade-off between the detail level and the precision of the predictions:

- The number of classes that need to be separated increases as one moves downwards in the ontology. The number of classes immediately below the root is

[1]Note that objects are also “moved” when discrimination between related classes is avoided. An object labeled with class c may, for example, be covered by the rules of a sub- or a superclass of c. However, this move is more conservative. The objects of c will only be covered by rules of the sub- or superclasses if the objects of these classes and the objects of c have similar information vectors.

[2]Recall that a supervised learning system makes predictions on the assumption that objects with similar information vectors have the same class. So, we are not introducing any new assumptions if we move an object x to a subclass where an object y has a similar information vector. We are just applying the same principle that makes prediction possible in the first place.

quite low, and it is therefore quite easy to predict these classes. However, the number of leaf classes is large (The process ontology, rev. 2.577 has 2, 725 leaf classes), and it may be very difficult to separate all of them. Thus, the difficulty of the learning problem increases with the detail level of the predictions.

- The number of objects is independent of the number of classes and will remain the same even if objects are moved. This means that the number of objects per class will decrease as we move downwards. So, not only must a larger number of classes be separated, but each class will have fewer objects that set it apart from the rest.

- The available information for discerning between the objects, i.e., the information vectors, remains the same as well. Hence, the ability to discern between the objects does not change, while more classes must be separated when objects are moved downwards. In particular, the information vectors may describe only properties of general classes, and it may not be possible to discern between detailed classes without additional data. The information vectors may consequently be insufficient for discerning between the more detailed classes.

Thus, it seems that it may not be possible to have both details and precision. There is a trade-off where the precision of the predictions increases as the detail level is reduced, and vice versa. In this trade-off, details should be sacrificed for precision since a classifier giving detailed, but inaccurate predictions is quite useless. However, no more details should be given away than what is absolutely necessary. Otherwise, we could just use the solution from Section 8.2.2 where only the most general classes were selected.

Note that there is also a biological reason for preferring more general classes. In Section 3.5.1 it was mentioned that similarly expressed genes may have different functions. They will therefore be annotated with different classes. If all of these classes are predicted for the similar expressed genes, many of the predictions will be incorrect. This may result in a considerable error rate. If a common superclass is predicted instead, these errors will not occur. Hence, moving genes upwards may also be a way of dealing with similarly expressed, but functionally dissimilar genes.

In the next two sections, we will introduce two different approaches that handle the problems that have just been discussed.

8.3 The ensemble method

A main concern for DAG learning is the scarcity of the data that is available for each class. The approach presented in this section attempts to avoid this problem by using all of the known objects (i.e., the objects labeled to a class c or the subclasses of c) when it learns rules for a class. At the same time, it avoids discriminating the known objects from the potential objects in the superclasses. The objects are moved upwards in other words, and the objects in related classes are not discerned from each other.

The method handles the trade-off between detail level and precision in two different ways. First, a "natural" balance between details and precision is created directly by the rules. The rules of most detailed classes will often be more specific than the rules of the superclasses. A previously unseen object is therefore more likely to satisfy a rule created for a general class than a rule created for a detailed class. Hence, an appropriate detail level may simply be found because the rules of the more specific classes will not cover the object.

Second, a voting procedure is applied on the rule predictions. The rules of detailed classes will often be less accurate than the rules of the general classes. So, noisy predictions to the detailed classes may occur. There is also a conflict between the rules. The objects of class c are covered not only by the rules of c, but also by the rules of the superclasses of c. The superclasses will consequently be predicted when c is predicted. So, a voting procedure is applied in order to remove the noisy predictions and resolve this conflict. It attempts to select a set of classes so that the loss of details and precision is minimized.

The method consists of a learning algorithm and a prediction algorithm, which are presented in the two next sections. It requires a DAG-decision systems with single decisions and expects that this system has been created from a DAG-decision with multiple decisions by the means of the transform that was given in Section 7.4. Moreover, it assumes that the DAG is well-defined. The simplified boundary set from Section 7.6.2 is therefore applied instead of the boundary sets defined in Section 7.6. Observe that these assumptions are made throughout this chapter.

8.3.1 Learning

The learning algorithm, which is given in Algorithm 8.1, finds two sets of rules for each class in the ontology except for the root. The first set, which is called the *consenting* set, predicts a class. The second set, which is called the *dissenting* set, predicts the complement of a class. Note that these two sets together form a binary classifier for a class. The full classifier may therefore be considered to consist of an ensemble of binary classifiers (cf. the name of the method).

When the rules of a class c is learned, the objects are divided into a positive set $\mathcal{P}$ containing the objects that should be covered by the rules and a negative set $\mathcal{N}$ containing the genes that should not be covered. The rules are then found by a subroutine **LearnRules**, which is described in the detail in Section 8.5. This is done twice so that both consenting and dissenting rules are learned for the class.

The definitions of $\mathcal{P}$ and $\mathcal{N}$ are crucial since these sets control the kind of rules that are made by the algorithm. In our case, these sets should fulfill several requirements.

1. The rules learned for class c should not discern the objects of c from the objects of the classes related to c.

2. The scarcity of the available data for a class c should be compensated by including all known objects (with regard to c) in $\mathcal{P}$ and not only objects annotated with c.

EnsembleLearner:
Input: A rooted DAG-decision system $\langle U, A, d, \succcurlyeq \rangle$ with root $\top$, and training accuracy γ.
Output: A set of rules RS.

```
 1: $RS = \emptyset$
 2: for all $c \in V_d - \{\top\}$ do
 3:     $\mathcal{P} = \overline{A}\,\overline{G}\,\overline{K}^*_c$
 4:     $\mathcal{N} = \underline{A}\,\underline{G}\,\underline{P}^*_{\sim\{c\}} \cap \overline{A}\,\underline{G}\,\underline{K}^*_{\sim\{c\}}$
 5:     $RS = RS \cup \text{LearnRules}(\mathcal{P}, \mathcal{N}, A, d, c, \gamma)$
 6:     $\mathcal{P}' = \overline{A}\,\underline{G}\,\underline{K}^*_{\sim\{c\}}$
 7:     $\mathcal{N}' = \underline{A}\,\overline{G}\,\overline{P}^*_c \cap \overline{A}\,\overline{G}\,\overline{K}^*_c$
 8:     $RS = RS \cup \text{LearnRules}(\mathcal{P}', \mathcal{N}', A, d, \overline{c}, \gamma)$
 9: end for
10: return RS
```

Algorithm 8.1: Learning in the ensemble approach

3. A gene may have several annotations, and all of these annotations should be predicted. This means that all objects in the transformed DAG-decision system should be covered by consenting rules.

4. The learning algorithm should create possible rules (with regard to $IND(A)$) for both the class and the complement such that conflicts in the boundary region can be considered by a voting system just as in an ordinary rough set approach.

The first two requirements can be fulfilled by using $\overline{K}^*_c$ for $\mathcal{P}$ and $\underline{K}^*_{\sim\{c\}}$ for $\mathcal{N}$ when rules are learned for c. When rules are learned for the complement of c, we may use $\underline{K}^*_{\sim\{c\}}$ for $\mathcal{P}'$ and $\overline{K}^*_c$ for $\mathcal{N}'$. However, the indiscernibility relations must be taken into account. There are two different indiscernibility relations: $IND(G)$, which is created by the multiple annotations, and $IND(A)$, which is induced by the attributes. In order to fulfill the third requirement, we need to assign a G-elementary set (i.e., a gene) to $\mathcal{P}$ (and $\mathcal{N}'$) if there is an object (i.e., an annotation) in this set that is known to belong to c. This is achieved by applying a G-upper approximation on $\overline{K}^*_c$ and a G-lower approximation on $\underline{K}^*_{\sim\{c\}}$. The last requirement can be satisfied by creating $\mathcal{P}$ and $\mathcal{P}'$ with an A-upper approximation and $\mathcal{N}$ and $\mathcal{N}'$ with an A-lower approximation.

The choice of upper approximations is a fairly simple as the elementary sets that are completely unknown[3] with regard to c (i.e., elementary sets that consist of only ob-

[3]The completely unknown sets consist of only objects that are labeled to the superclasses of c. These sets are included in $\overline{B}_{\underline{L}^*}(c)$ and $\overline{B}_{\overline{L}^*}(c)$. However, if these approximations were applied, the

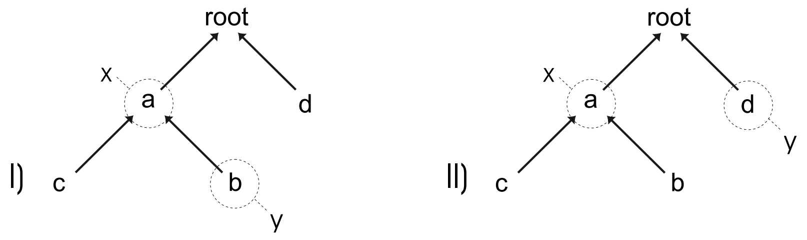


Figure 8.3: *Two situations when a* $\sim\{c\}$*-likely-known may occur.* x and y are objects and a, b, c, and d are classes. c is the class that we would like to learn.

jects labeled to the superclasses of c) should not be included in these approximations. So only following approximations may be applied:

- $\overline{B}_{\underline{K}^*}(\sim\{c\}) = \overline{B}\,\underline{K}^*_{\sim\{c\}}$
- $\overline{B}_{\overline{K}^*}(c) = \overline{B}\,\overline{K}^*_c$

For the lower approximations, these operators may be used:

- $\underline{B}_{\underline{K}^*}(\sim\{c\}) = \underline{B}\,\underline{K}^*_{\sim\{c\}}$
- $\underline{B}_{\overline{K}^*}(c) = \underline{B}\,\overline{K}^*_c$
- $\underline{B}_{\underline{L}^*}(\sim\{c\}) = \underline{B}\,\underline{P}^*_{\sim\{c\}} \cap \overline{B}\,\underline{K}^*_{\sim\{c\}}$
- $\underline{B}_{\overline{L}^*}(c) = \underline{B}\overline{P}^*_c \cap \overline{B}\,\overline{K}^*_c$

The difference between these are that the last two contain $\sim\{c\}$-likely-known sets or $\{c\}$-likely-known sets while the first two do not. So the question is whether $\sim\{c\}$-likely-known sets (or $\{c\}$-likely-known) should be included. It appears that the indiscernibility relations behave differently with regard to this question so that $\sim\{c\}$-likely-known sets should be included in the A-lower approximations, but not in G-lower approximations. This is explained in the following example.

Example 8.2. Figure 8.3 illustrates the two main situations where a $\sim\{c\}$-likely-known may occur. We are trying to learn examples for c. Object x is labeled to a superclass of c, while object y belongs to a complement class of c.

Assume x and y belong to different G-elementary sets, but to the same A-elementary set. $[y]_A$ is therefore a $\sim\{c\}$-likely-known set with regard to $IND(A)$. In the first

completely unknown sets would be covered by the rules of c and the rules of any sibling of c (i.e., an unrelated class that has same immediate superclass as c). This means c and all of its siblings would be predicted for the unknown objects, and we would not obtain any more information than we would if a superclass was predicted instead. Consequently, it is better that these objects are only covered by the rules of superclasses.

case in Figure 8.3, y is labeled to a subclass of a. It is therefore more likely that x belongs to b than it belongs to c. More importantly, the algorithm will not try to discern y from the objects in $\overline{K}_c^*$ if $[y]_A$ is not assigned to $\mathcal{N}$. This means that the rules created for c may actually cover y such that c may be predicted for y. This will obviously lead to incorrect predictions. Hence, $[y]_A$ should be assigned to $\mathcal{N}$, which is achieved by creating $\mathcal{N}$ with $\underline{A}_{\underline{L}^*}(\sim\{c\})$.

In the second case in Figure 8.3, the classes of x and y are not related. x could perhaps belong to c or b. However, the class labels of x and y are inconsistent, and if the rules of c and b are allowed to cover x, this conflict may actually become larger. It is therefore better to assign $[y]_A$ to $\mathcal{N}$ also in this case.

The situation is quite different, however, if x and y belong to the same G-elementary set. x and y represent two different annotations for the same gene in this case. Hence, there is no conflict between them since both of them should be predicted (cf. the third requirement). The rules of c (or b) should therefore be allowed to cover x in the second case in Figure 8.3 since x could belong to this subclass. This means that $[y]_G$ should not be assigned to $\mathcal{N}$.

This may also hold in the first case in Figure 8.3. However, this depends on the interpretation that we apply to the annotations. In this case, x and y are labeled to related classes, which is really a peculiar way of annotating a gene. We may assume that x (i.e., the annotation to a) belongs to b, but this means that x is redundant since y is already labeled b. So x should in fact have been removed. Alternatively, we may assume that x is an indication that the gene may belong to another subclass of a besides b. We choose this interpretation as this seems to be more reasonable (and creates the simplest approximations). Hence, $[y]_G$ should not be assigned to $\mathcal{N}$, which is achieved with $\underline{G}_{\underline{K}^*}(\sim\{c\})$. ■

Thus, $\underline{G}_{\underline{K}^*}(\sim\{c\})$ should be used as G-lower approximation, while $\underline{A}_{\underline{L}^*}(\sim\{c\})$ (and $\underline{A}_{\overline{L}^*}(\sim\{c\})$) should be applied as A-lower approximation. The approximations are applied in two steps[4]. The G-approximations are first applied to $\overline{K}_c^*$ and $\underline{K}_{\sim\{c\}}^*$. This creates the two sets $\overline{G}\,\overline{K}_c^*$ and $\underline{G}\,\underline{K}_{\sim\{c\}}^*$ ($\overline{G}\,\overline{P}_c^*$ and $\underline{G}\,\underline{P}_{\sim\{c\}}^*$ are also created from $\overline{P}_c^*$ and $\underline{P}_{\sim\{c\}}^*$). The A-approximations are then applied to these sets. This done by substitution so that $\overline{K}_c^*$ is replaced by $\overline{G}\,\overline{K}_c^*$, and $\underline{K}_{\sim\{c\}}^*$ is replaced by $\underline{G}\,\underline{K}_{\sim\{c\}}^*$, etc. For example, $\overline{A}_{\overline{K}^*}(c) = \overline{A}\,\overline{K}_c^*$ becomes $\overline{A}\,\overline{G}\,\overline{K}_c^*$. This creates the four sets that are used in the algorithm:

- $\mathcal{P} = \overline{A}\,\overline{G}\,\overline{K}_c^*$ (from $\overline{B}_{\overline{K}^*}(c)$)
- $\mathcal{N} = \underline{A}\,\underline{G}\,\underline{P}_{\sim\{c\}}^* \cap \overline{A}\,\underline{G}\,\underline{K}_{\sim\{c\}}^*$ (from $\underline{B}_{\underline{L}^*}(\sim\{c\})$)
- $\mathcal{P}' = \overline{A}\,\underline{G}\,\underline{K}_{\sim\{c\}}^*$ (from $\overline{B}_{\underline{K}^*}(\sim\{c\})$)
- $\mathcal{N}' = \underline{A}\,\overline{G}\,\overline{P}_c^* \cap \overline{A}\,\overline{G}\,\overline{K}_c^*$ (from $\underline{B}_{\overline{L}^*}(c)$)

[4]This means that we first decide whether the genes belong to a class or the complement (using the G-approximations) and then consider whether the genes have similar information vectors (using the A-approximations).

EnsemblePredictor:
Input: A G-elementary set $[x]_G$, a set of rules RS, a partial order $\langle V_d, \succcurlyeq \rangle$, and a threshold θ.
Output: A set P of class predictions for $[x]_G$.

```
1:  $RS' = \{(\alpha \to \beta) \in RS \mid [x]_G \subseteq [\![\alpha]\!]_{\mathcal{A}}\}$
2:  $Q = \{c \in V_d \mid (\alpha \to \langle d, c \rangle) \in RS'\}$
3:  for all $c \in Q$ do
4:      $RS'_c = \{(\alpha \to \beta) \in RS' \mid \beta = \langle d, c \rangle\}$
5:      $RS'_{\overline{c}} = \{(\alpha \to \beta) \in RS \mid \beta = \langle d, \overline{c} \rangle\}$
6:      $SA(c) = \sum_{r \in RS'_c} SA(r)$
7:      $SC(c) = \sum_{r \in RS'_c} SC(r)$
8:      $WB(c) = \sum_{r \in RS'_c} WB(r)$
9:      $SN(c) = \sum_{r \in RS'_{\overline{c}}} SN(r)$
10: end for
11: $(P, PGain, EstWB) = SelectClasses(\top, Q, \langle V_d, \succcurlyeq \rangle, \theta)$
12: return $(P)$
```

Algorithm 8.2: Prediction in the ensemble approach

Observe that we do not set an absolute restriction on the coverage of the rules. The coverage $Cov_{\mathcal{A}}(RS_c)$ of the rules for c are rather defined by a lower limit $\overline{A}\,\overline{G}\,\overline{K}^*_c$ and an upper limit $\overline{A}\,\overline{G}\,\overline{K}^*_c \cap \underline{A}\,\overline{G}\,\overline{P}^*_c$ (i.e., $\mathcal{P} = \overline{A}\,\overline{G}\,\overline{K}^*_c \subseteq Cov_{\mathcal{A}}(RS_c) \subseteq \overline{A}\,\overline{G}\,\overline{K}^*_c \cap \underline{A}\,\overline{G}\,\overline{P}^*_c = U - \underline{A}\,\underline{G}\,\underline{P}^*_{\sim\{c\}} \cap \overline{A}\,\underline{G}\,\underline{K}^*_{\sim\{c\}} = U - \mathcal{N}$). This means that the rules are allowed to cover objects annotated to the superclasses if this makes them simpler. Hence, the classifier may actually give more detailed predictions than the original annotations. However, this happens only when the known objects of c and the objects of the superclasses have something in common. The rule learning algorithm will not try to create rules that just cover objects in the superclasses.

8.3.2 Prediction

The consenting rules of a class cover all objects that are known to belong to the class. An object labeled to class c will therefore be covered by rules of c and by the rules of any superclass of c. This means that all of these classes will be predicted for the object, and one may get many redundant predictions. We resolve this issue with a voting procedure, which selects a set of classes so that the loss of details and precision is minimized.

This procedure consists of two parts shown in Algorithms 8.2 and 8.3 where the latter is called by the former. Algorithm 8.2 takes a G-elementary set[5] and selects the

[5]Predictions are made for G-elementary sets rather than for objects since the G-elementary sets

rules in RS that cover this set. Then, it collects the classes predicted by these rules and stores them in Q. The votes for each class in Q are computed, and the procedure **SelectClasses** is called so that a subset of Q is selected.

Votes

Four different kinds of votes are maintained for each class c in Q: $SA(c)$, $SC(c)$, $WB(c)$, and $SN(c)$. These are computed from the rules in RS'. The first three are computed from the consenting rules, while the last is computed from the dissenting rules. A consenting rule $r = (\alpha \rightarrow \langle d, c\rangle)$ maintains the following kinds of votes:

- **The support above** c: $SA(r)$ is the number of objects that are labeled to superclasses of c and covered by r.

$$SA(r) = |[\![\alpha]\!]_{\mathcal{A}} \cap (\overline{P}^*_c - \overline{K}^*_c)|$$

- **The support of** c: $SC(r)$ is the number of objects that are labeled to c and covered by r.

$$SC(r) = |[\![\alpha]\!]_{\mathcal{A}} \cap [\![\langle d, c\rangle]\!]_{\mathcal{A}}|$$

- **The weighted support below (and of)** c: $WB(r)$ is a weighted sum over the objects that are covered by r and labeled to c or one of subclasses of c.

$$WB(r) = \sum_{x \in ([\![\alpha]\!]_{\mathcal{A}} \cap \overline{K}^*_c)} p(c, d(x))$$

  Note that $WB(r)$ corresponds to the number of objects that are known to belong to c and covered by r if $p(c, e) = 1$ for all $e \in V_d$ (where $c \succcurlyeq e$).

A dissenting rule $r = (\alpha \rightarrow \langle d, \overline{c}\rangle)$ has the following kinds of votes:

- **The negative support of** c: $SN(r)$ is number of G-elementary sets in $\underline{G}\,\underline{K}^*_{\sim\{c\}}$ that are covered by r.

$$SN(r) = |\{[x]_G \mid [x]_G \subseteq ([\![\alpha]\!]_{\mathcal{A}} \cap \underline{K}^*_{\sim\{c\}})\}|$$

The function $p(c, e)$, which is used by $WB(r)$, defines a penalty when a prediction to c does not match the original detail level of an object. We assume that $p(c, e)$ has a range between 0 and 1 when $c \succcurlyeq e$ and is equal to 1 when $c = e$. Moreover, we

correspond to the objects (i.e., the genes) in the original DAG-decision system with multiple decisions. It is the class(es) of these objects (i.e., the genes) that should be predicted. Note that this difference is rather technical as all of the objects of a G-elementary set have the same information vector. So, if a rule covers one object of a G-elementary set, it must cover the rest of the objects as well.

assume that $p(c, e)$ can be decomposed as $\frac{f(c)}{f(e)}$ where the function f describes some property of the class.

There are many functions that may be chosen as penalty function. One possibility would be to consider the ratio between the average path length as a measure of how well c reassembles e.

$$p_1(c, e) = \frac{apl(c)}{apl(e)} \qquad \text{where} \qquad apl(c) = \frac{1}{|Paths(c, \top)|} \sum_{t \in Paths(c,\top)} ||t||$$

A similar measure can be designed by counting the number of classes that are above c and e.

$$p_2(c, e) = \frac{|[c]^{\succcurlyeq}|}{|[e]^{\succcurlyeq}|}$$

Alternatively, $f(c) = 1/|[c]^{\preccurlyeq}|$ could be used as measure of the detail level of c. A class with many subclasses would, in this case, be considered to be very general and have a low $f(c)$. The penalty function could then be defined as:

$$p_3(c, e) = \frac{1/|[c]^{\preccurlyeq}|}{1/|[e]^{\preccurlyeq}|} = \frac{|[e]^{\preccurlyeq}|}{|[c]^{\preccurlyeq}|}$$

In our current implementation, we use:

$$p_i(c, e) = \frac{\max\limits_{t_c \in Paths(c,\top)} ||t_c||}{\max\limits_{t_e \in Paths(e,\top)} ||t_e||}$$

where t_c must be a subpath of t_e ($t_c \sqsubseteq t_e$). It is quite similar to the $p_1(c, e)$ and $p_2(c, e)$.

Remark 1. The support measures ($SA(c), SC(c)$, and $WB(c)$) for the consenting votes count objects, while the support measure ($WB(c)$) for the dissenting votes, counts G-elementary sets. Hence, the definitions of votes cast for and against a class may appear to be asymmetrical. However, this is not a mistake. The support measures are supposed to mirror the global recall RA and the global precision RP, which will be defined in Section 9.2. The recall measures the share of the annotations that are matched by the predictions, and the precision assesses the share of the predictions that match an annotation. The definitions of the votes just reflect the impact a prediction to a class c will have on the RA and RP. A prediction to c may match several annotations of the same gene, but it will only add one incorrect prediction for each gene that has no annotations related to c. Hence, objects must be counted for votes in favor of a class, and G-elementary set must be counted for the votes against a class.

Selection of classes

The procedure **SelectClasses** in Algorithm 8.3 applies the votes to select a subset of the classes in Q. It starts at the root ($\top$) and traverses the DAG depth first. At each class, it decides whether this class should be selected or not. Initially, it examines if the class has been visited before and avoids recomputing its decision for the class if the class has been marked as visited. Instead it just looks up the result in an array T (line 33).

If the class is visited for the first time, on the other hand, it visits the subclasses and makes a decision for each of them. These decisions are passed up to the class and merged in lines 4-10. We will come back to the details of this process. However, the outcome of this process is a set S, which contains the selected subclasses, and the total gain $SGain$, which is associated with these classes. $SGain$ is sum of the gain of classes in S, i.e.,

$$SGain = \sum_{e \in S} Gain(e)$$

The gain associated with a class e is defined as the votes cast for e (except for the support above of e) minus the votes cast against e.

$$Gain(e) = WB(e) - SN(e)$$

After the decisions have been merged, the algorithm determines if the class is among the selectable classes in Q (line 11) and just returns the merged decisions for the subclasses (line 29) if it is not. If the class is selectable, on the other hand, it decides whether the class should be selected. This is done in two different ways depending on contents of S:

- If S is empty (such that no subclasses have been selected or there are no subclasses), the algorithm has to decide whether the class should be selected or not. Thus, it compares the votes cast for the class with the votes cast against it (line 13). If there are more votes for the class, it is added to P (such that it is selected). If there are more votes against the class, P is set to the empty set (such that no class is selected).

- If S is not empty such that some subclasses have been selected, the algorithm has to choose between the class c or the selected subclasses. It makes its decision by comparing the gain of c with the gain of subclasses in S. This comparison is made in line 20. The condition in this **if**-statement may seem complicated. However, variable $SEstWB$ is just a correction for $(WB(c) - SC(c))$ and may be ignored for now. The condition can therefore be simplified to

  $$WB(c) - SC(c) - SN(c) > SGain$$

  This condition may again be rewritten as

  $$Gain(c) > SGain + SC(c)$$

  where term $SC(c)$ is the support of c. This term is already included in $WB(c)$ and $Gain(c)$, but it is not in $SGain$. However, the subclasses in S are suitable

SelectClasses
Input: A class c, a set of predictable classes Q, a partial order $\langle V_d, \succcurlyeq \rangle$, and a threshold θ
Output: A set of classes P, their gain $PGain$, and estimated votes $EstWB$

```
1: if class $c$ has not been visited then
2:    Mark $c$ as visited
3:    $S = \emptyset$, $SGain = 0$, $B = \emptyset$, $BGain = 0$, $SEstWB = 0$
4:    for all $e \in Sub(c)$ do {For all immediate subclasses of $c$}
5:       $(S', SGain', SEstWB') = \text{SelectClasses}(e, Q, \langle V_d, \succcurlyeq \rangle, \theta)$
6:       if $SGain > 0$ then $S = S \cup S'$, $SGain = SGain + SGain'$
7:       if $B = \emptyset$ or $(S' \neq \emptyset$ and $SGain' > BGain)$ then $B = S'$, $BGain = SGain'$
8:       $SEstWB = SEstWB + SEstWB'$
9:    end for
10:   if $S = \emptyset$ then $S = B$, $SGain = BGain$
11:   if $c \in Q$ then {Is $c$ among the predictable classes?}
12:      if $S = \emptyset$ then {Has any of the subclasses been selected?}
13:         if $SA(c) + WB(c) \geq SN(c)$ then
14:            $P = \{c\}$, $PGain = WB(c) - SN(c)$ {$c$ is selected}
15:         else
16:            $P = \emptyset$, $PGain = 0$ {no class is selected}
17:         end if
18:         $EstWB = 1/f(c) \cdot WB(c)$
19:      else
20:         if $(1-\theta)(WB(c) - SC(c)) + \theta \cdot f(c) \cdot SEstWB - SN(c) > SGain$ then
21:            $P = \{c\}$, $PGain = WB(c) - SN(c)$ {$c$ is selected}
22:            $EstWB = 1/f(c) \cdot WB(c)$
23:         else
24:            $P = S$, $PGain = SGain + SC(c)$ {The subclasses are selected}
25:            $EstWB = SEstWB + 1/f(c) \cdot SC(c)$
26:         end if
27:      end if
28:   else
29:      $P = S$, $PGain = SGain$, $EstWB = SEstWB$
30:   end if
31:   Store $T[c] = (P, PGain, EstWB)$
32: else
33:   Look up $(P, PGain, EstWB) = T[c]$
34: end if
35: return $(P, PGain, EstWB)$
```

Algorithm 8.3: Selection of classes in the ensemble approach

as predictions for the objects that are labeled to c. Therefore, $SC(c)$ is added to $SGain$ so that this support is also included in the gain of S.

Hence, the algorithm compares the gain of c with the gain of subclasses in S. It selects c if it has the largest gain, and the subclasses otherwise. It then assigns its selection to P and computes $PGain$ accordingly.

Remark 2. The support above c, $SA(c)$, is included in neither the gain nor the comparison in line 20. The reason is that both c and any of the subclasses in S may serve as predictions for these objects. Hence, these votes must be assigned to both c and S and will not contribute to select either one. Moreover, if this support is included in the gain it will occur in the individual gain of all subclasses in S. It will therefore be repeated in $SGain$ for each of these subclasses, and this repeated support will have to be removed during the merge process. Hence, it is easier to leave this support out of the gain and the comparison in line 20.

We may now explain the details of the merge process in lines 4-10. It basically collects all selections that have been made at the subclasses of c and stores them in S. However, a subclass e may have been selected in line 13 because it has a large $SA(e)$. Its gain[6] may actually be negative so that it would not have been selected without the support of the superclasses. All subclasses of c share the votes that are above c. Hence, several of these classes may have been selected on the basis of the same votes and may not have the support on their own to be selected. Since any subclass of c may be considered as a correct prediction for the objects that are labeled to the superclasses of c, it is not necessary to include all of them. They will only reduce the total gain of the subclasses since their individual gain is negative. The highest gain is in fact achieved by accepting only the selections that have a positive gain or the one that has the highest gain if no selection has a positive gain.

Hence, the algorithm examines each selection made at the subclasses and only adds a selection to S if it has a positive gain. At the same time, it keeps track of the selection that has the highest gain and stores this selection in B (and its gain in $BGain$). Afterwards, it determines if every selection had a negative gain by testing if S is empty (line 10) and assigns B to S if this is the case.

Example 8.3. Assume that we want to predict classes for a G-elementary set $[x]_G$, and the algorithm in Algorithm 8.2 has computed the votes that are given in Figure 8.4. **SelectClasses** will, in this case, start at root and move down to c. It will select c since $SA(c) + WB(c) = 11 \geq 1 = SN(c)$. It will then move to d and b, but it will not consider these classes since they have not been predicted. The gain of c will just be passed to a. The algorithm will consider f next. This class will be selected since $SA(f) + WB(f) = 12 \geq 5 = SN(f)$. However, g will not be selected as $SA(g) + WB(g) = 8 < 10 = SN(g)$. At e, the algorithm has to choose between f and this class. The gain of e is $WB(e) - SN(e) = 13$, while the gain of f is $WB(f) - SN(f) + SC(e) = 6$. Hence, e will be chosen. The algorithm will then move

[6]Note that the condition in line 13 can be written as $SA(e) + Gain(e) \geq 0$.

Class	***SA***	***SC***	***WB***	***SN***
a	0	1	10.5	0
b	Not predicted by the rules			
c	1	10	10	1
d	Not predicted by the rules			
e	1	3	15	2
f	4	8	8	5
g	4	4	4	10
h	0	3	6	5
i	3	4	4	5
j	3	2	2	5

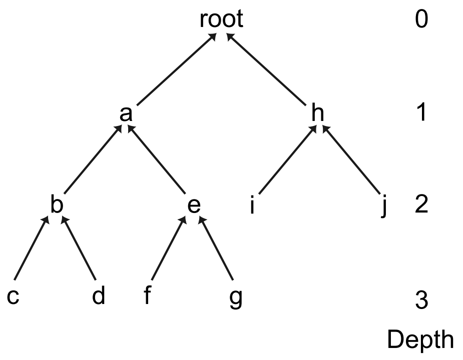


Figure 8.4: *An example of prediction with* ***SelectClasses***. The table displays the votes that have been computed from the rules. The classes in the table refer to the ontology on the right. WB is computed with penalty function $p(c,e) = depth(c)/depth(e)$.

to a and consider if this class should be selected instead of c and e. It will pick the subclasses since the gain of a is $WB(a) - SN(a) = 10.5$, and the gain of c and e is $(WB(c) - SN(c)) + (WB(e) - SN(e)) + SC(a) = (10 - 1) + (15 - 2) + 1 = 23$.

After **SelectClasses** has visited a, it will consider h and its subclasses. Both i and j will be selected initially since $SA(i)+WB(i) = 7 \geq 5 = SN(i)$ and $SA(j)+WB(j) = 5 \geq 5 = SN(j)$. However, both of them have a negative gain $WB(i) - SN(i) = -1$ and $WB(j) - SN(j) = -3$ so that j will be removed during the subclass merger at h. The algorithm will therefore consider only i when it decides if h should be selected. In this case, it will choose i since the gain of h is $WB(h) - SN(h) = 6 - 5 = 1$, and the gain of i is $WB(i) - SN(i) + SC(i) = 4 - 5 + 3 = 2$. Hence, its prediction will be c, e, and i, which are the classes with the largest support in favor of them and the least support against them. ■

A corrective to *WB(c)*

Two different estimates of the support below c are applied in the condition in line 20. $WB(c)-SC(c)$, which is computed from the rules that predicts c, and $f(c)\cdot SEstWB$, which is computed from the SC support as the algorithm moves through the ontology. The reason is that rules in the more general classes may overestimate the WB support, and this has the effect that too general classes may be predicted if only $WB(c)-SC(c)$ is used.

Example 8.4. Assume that we have two objects x and y with information vectors (and that these objects are the sole members of their respective G-elementary set):

$$Inf_A(x) = \{\, \langle \texttt{0H-15m, down} \rangle, \langle \texttt{15m-30m, down} \rangle, \langle \texttt{30m-1H, down} \rangle, \langle \texttt{1H-2H, up} \rangle, \langle \texttt{2H-4H, up} \rangle \}$$

$$\begin{aligned} Inf_A(y) = \{\, & \langle \texttt{0H-15m, down} \rangle, \langle \texttt{15m-30m, down} \rangle, \langle \texttt{30m-1H, down} \rangle, \langle \texttt{1H-2H, const} \rangle, \\ & \langle \texttt{2H-4H, const} \rangle \} \end{aligned}$$

Moreover, x is labeled to the class c and y is labeled to the class d in the ontology in Figure 8.4. The learning algorithm may, in this case, create the following rules for c, d, and b (their common superclass):

$$\begin{array}{lll} \langle \texttt{15m-30m, down} \rangle \wedge \langle \texttt{30m-1H, down} \rangle \wedge \langle \texttt{1H-2H, up} \rangle & \rightarrow \langle \texttt{Class, c} \rangle & (WB(c) = 1) \\ \langle \texttt{15m-30m, down} \rangle \wedge \langle \texttt{30m-1H, down} \rangle \wedge \langle \texttt{1H-2H, const} \rangle & \rightarrow \langle \texttt{Class, d} \rangle & (WB(d) = 1) \\ \langle \texttt{15m-30m, down} \rangle \wedge \langle \texttt{30m-1H, down} \rangle & \rightarrow \langle \texttt{Class, b} \rangle & (WB(b) = 4/3) \end{array}$$

where their WB support is computed with $p(c,e) = depth(c)/depth(e)$. The rule for b covers both objects and has a higher WB support than the other rules. This means that b will have a higher gain than c and d if each class has a negative support of 0. The algorithm will therefore predict b for x and y. However, the more detailed classes c and d should be predicted in this case since the information vectors of x and y are discernible, and the negative support of c and d is the same as the negative support of b. ■

The problem in the last example is that a rule of the superclass b covers two different information vectors that belong to different subclasses. Its WB support is therefore too high compared to the WB support of the subclasses since it does not discern between these vectors. The SC support is less affected by this problem since it only counts the objects that are labeled to the class. The problem may therefore avoided if the support below c can be estimated from the SC support of the subclasses. This is possible since $WB(c)$ can be decomposed as follows:

$$WB(c) = \sum_{(\alpha\rightarrow\beta)\in RS'_c} \sum_{x\in([\![\alpha]\!]_{\mathcal{A}}\cap\overline{K}^*_c)} p(c,d(x)) \approx \sum_{x\in(\overline{K}^*_c\cap T_c)} p(c,d(x))$$

where $T_c = \bigcup_{(\alpha\rightarrow\beta)\in RS'_c} [\![\alpha]\!]_{\mathcal{A}}$. $\overline{K}^*_c$ is equal to $\bigcup_{c\succnsim e} [\![\langle d,e\rangle]\!]_{\mathcal{A}}$ and the $[\![\langle d,e\rangle]\!]_{\mathcal{A}}$-sets are disjoint since each objects is labeled to a single class. Hence,

$$\begin{aligned} \sum_{x\in\overline{K}^*_c\cap T_c)} p(c,d(x)) &= \sum_{c\succnsim e} \sum_{x\in T_c\cap[\![\langle d,e\rangle]\!]_{\mathcal{A}}} p(c,d(x)) \\ &= \sum_{c\succnsim e} p(c,e)\cdot|T_c\cap[\![\langle d,e\rangle]\!]_{\mathcal{A}}| \\ &\approx \sum_{c\succnsim e} p(c,e) \sum_{(\alpha\rightarrow\beta\in RS'_e)} |[\![\alpha]\!]_{\mathcal{A}}\cap[\![\langle d,e\rangle]\!]_{\mathcal{A}}| \\ &= \sum_{c\succnsim e} p(c,e)\cdot SC(e) \end{aligned}$$

$WB(c) - SC(c)$ may therefore be computed as

$$WB(c) - SC(c) \approx \sum_{c\succ e} p(c,e)\cdot SC(e) = f(c)\sum_{c\succ e}\frac{1}{f(e)}SC(e) = f(c)\cdot SEstWB$$

where $SEstWB = \sum_{c \succ e} \frac{1}{f(e)} SC(e)$.

$SEstWB$ is computed by the algorithm as it moves through the subclasses. The estimates from the subclasses of c are added up in line 8, and the support of the classes is added to $EstWB$ in line 25 (when the subclasses are selected). However, $1/f(c) \cdot SC(c)$ is not added to $EstWB$ in the lines 18 and 22. The reason is that $EstWB$ will be compared to the corresponding $PGain$, which is based on $WB(c)$. So $1/f(c) \cdot WB(c)$ is assigned to $EstWB$ such that these quantities correspond.

Unfortunately, the estimate $SEstWB$ is not without flaws. The poor rules in the subclasses of c may give noisy predictions, which may lead to estimation error in $SEstWB$. $WB(c)$, on the other hand, is based on the rules of c and will be less affected by such errors. Therefore, a combination of $WB(c) - SC(c)$ and $f(c) \cdot SEstWB$ is used in line 20 where the threshold θ controls the influence of each estimate. The decision of the algorithm will be based entirely on $f(c) \cdot SEstWB$ if $\theta = 1$ — in which case the algorithm may produce erroneous predictions — and on $WB(c) - SC(c)$ if $\theta = 0$ — in which case the algorithm may create too general predictions. Hence, θ may be used to control the detail level and the precision of the predictions.

8.3.3 An approximate approach

Algorithm 8.1 creates two sets of rules for each class. One is a consenting set, which is created such that the votes in favor of a class can be estimated. The other set, is a dissenting set, which is made such that the votes against the class can be estimated. The dissenting set would not be required if the negative support of a class could be estimated from the consenting rules of its complement classes. A more efficient algorithm could therefore be designed since it would only need to learn a consenting set for each class. Unfortunately, it is not possible to compute the negative support precisely from the consenting rules (as will be explained shortly). Only an approximate estimate may be obtained. Still, an approximate version of the ensemble method may work well in practice, and such an approach is therefore introduced in this section.

The negative support of a rule $r = (\alpha \to \langle d, \overline{c} \rangle)$ in the dissenting set is defined as:

$$SN(r) = |\{[x]_G \mid [x]_G \subseteq (\llbracket \alpha \rrbracket_{\mathcal{A}} \cap \underline{K}^*_{\sim\{c\}})\}|$$

It counts basically the number of G-elementary sets in $\underline{G}\,\underline{K}^*_{\sim\{c\}}$ that are covered by the rule. So, if we want to estimate the negative support from the consenting sets, we must determine how many G-elementary set in $\underline{G}\,\underline{K}^*_{\sim\{c\}}$ that are covered by similar consenting rules.

It is easily seen that the consenting rules of classes in $\sim\{c\}$ will cover the objects that are in $\underline{G}\,\underline{K}^*_{\sim\{c\}}$. The consenting rules of a class e cover the objects in $\overline{A}\,\overline{G}\,\overline{K}^*_e$ and it can be shown by the means of Definition 7.20 and Property 3 in Section 4.3 that

$$\bigcup_{e \in \sim\{c\}} \overline{A}\,\overline{G}\,\overline{K}^*_e = \overline{A}\,\overline{G} \left(\bigcup_{e \in \sim\{c\}} \overline{K}^*_e \right) = \overline{A}\,\overline{G}\,\overline{K}^*_{\sim\{c\}} \supseteq \underline{G}\,\underline{K}^*_{\sim\{c\}}$$

So, it is possible that the negative support could be estimated from these classes. However, in order to estimate the support, we must first assign a new support measure to the consenting rules. The number of G-elementary sets in $\underline{G}\,\underline{K}^*_{\sim\{c\}}$ must then be computed as sum of the support of the classes in $\sim\{c\}$. This means that $\underline{G}\,\underline{K}^*_{\sim\{c\}}$ must be represented as a union of disjoint sets that are associated with the classes in $\sim\{c\}$.

We begin by transforming $\underline{K}^*_{\sim\{c\}}$ such that it is expressed as a union. There are several possibilities in this case. Definition 7.20 states that

$$\underline{K}^*_{\sim\{c\}} = \bigcup_{e \in \sim\{c\}} \underline{K}^*_e$$

So, one possibility would be to defined the support on $\underline{K}^*_e$. However, these set are not disjoint (i.e., $\underline{K}^*_{e_1} \cap \underline{K}^*_{e_2}$ may not be empty for $e_1, e_2 \in \sim\{c\}$). So, we cannot use all of them and must select some. One possible choice would be to select sets that are associated with the most general classes in $\sim\{c\}$.

$$E_1 = \bigcup_{e \in Z_{mgc}} \underline{K}^*_e \quad \text{where} \quad Z_{mgc} = \{e \in \sim\{c\} \mid \forall f \in \sim\{c\}\, f \not> e\}$$

Another would be to use the most specific classes in $\sim\{c\}$.

$$E_2 = \bigcup_{e \in Z_{msc}} \underline{K}^*_e \quad \text{where} \quad Z_{msc} = \{e \in \sim\{c\} \mid \forall f \in \sim\{c\}\, e \not> f\}$$

However, these unions will be only subsets of $\underline{K}^*_{\sim\{c\}}$ so that the estimate will not be exact in this case. Alternatively, we may transform $\underline{K}^*_{\sim\{c\}}$ with the aid of Theorem 7.5a and Lemma 7.2a as follows:

$$\underline{K}^*_{\sim\{c\}} = \underline{K}_{\sim\{c\}} = U - \overline{P}_{\{c\}} = \{x \in U \mid c \not\approx d(x)\} = \bigcup_{e \not\approx c} X_e = \bigcup_{e \in \sim\{c\}} X_e$$

where $X_e = \{x \in U | d(x) = e\}$. These sets are pairwise disjoint (i.e, $X_{e_1} \cap X_{e_2} = \emptyset$, for $e_1, e_2 \in V_d$). So, an exact solution may be obtained with

$$E_3 = \bigcup_{e \in \sim\{c\}} X_e$$

Still, we need to count the G-elementary sets in $\underline{G}\,\underline{K}^*_{\sim\{c\}}$ and must apply the G-lower approximation to the estimates for $\underline{K}^*_{\sim\{c\}}$. Unfortunately, the lower approximation does not maintain unions. Property 4 in Section 4.3 tells us that only

$$\underline{G}\left(\bigcup_{e \in Z} Y_e\right) \supseteq \bigcup_{e \in Z} \underline{G}Y_e$$

holds. This means that an exact solution cannot be obtained regardless which estimate is used for $\underline{K}^*_{\sim\{c\}}$. Only an underestimate of the negative support may be computed from the consenting rules.

The upper approximation, on the other hand, maintains unions of sets (property 3 in Section 4.3) such that

$$\overline{G}\left(\bigcup_{e\in Z} Y_e\right) = \bigcup_{e\in Z} \overline{G}Y_e$$

An exact estimated could be obtained if this approximation could been used. However, the G-upper approximation may contain G-elementary sets that belong partially to c or its related classes, and these sets should not be counted. The upper approximation also creates sets that are not disjoint so that the same G-elementary set may be counted several times. This means that the upper approximation will provide a large overestimate of the negative support.

Hence, neither the lower nor the upper approximation gives an accurate estimate. The lower approximation underestimates the negative support, while the upper approximation overestimates it. A combination of these two may still provide a good estimate as the exact negative support lies between them. This is the idea behind the approach presented here. Two support scores are associated with each consenting rule: $SU(r)$, which is based on the upper approximation and $SL(r)$, which is based on the lower approximation. The negative support of a rule is then computed as

$$(1-t)\cdot SU(r) + t\cdot SL(r)$$

where t is a threshold that controls the influence of each estimate (This is currently set to 0.5 in our implementation).

Still, we have to decide on an estimates of $\underline{K}^*_{\sim\{c\}}$ that should be used. We have actually tried all of them. E_3 seems most appealing since this is an exact estimate. However, the estimate provided by the lower approximation is quite small in this case since only G-elementary sets with one object may occur in $\underline{G}X_e$. So, our success with this estimate was only limited[7]. E_2 did not provide a good estimate, either. It seemed to overestimate the negative support such that too general classes were predicted. The best results were obtained by selecting the most specific classes that were predicted and estimate the support from them. This estimate is therefore used in the approach that is presented here.

This approach is shown in Algorithms 8.4 and 8.5. Note that Algorithm 8.4 is basically the same algorithm as Algorithm 8.4 except that is does not learn dissenting rules. Algorithm 8.5 is also quite similar to Algorithm 8.2. The main difference is that SN(C) is now computed from consenting rules of the most specific classes in $\sim\{c\}\cap Q$. The upper and the lower supports are defined as

$$SU(r) = |\{[x]_G \mid [x]_G \cap [\![\alpha]\!]_{\mathcal{A}} \cap \neq \underline{K}^*_e \emptyset\}|$$
$$SL(r) = |\{[x]_G \mid [x]_G \subseteq ([\![\alpha]\!]_{\mathcal{A}} \cap \underline{K}^*_e)\}|$$

[7]Note that our time for experimenting with this approach was very limited. It is possible that a good estimate may be obtained with both E_2 and E_3 by tuning the threshold t. However, we did not achieve satisfactory results with these estimates in the time we had available.

```
EnsembleLearner:
Input: A rooted DAG-decision system $\langle U, A, d, \succcurlyeq \rangle$ with root $\top$, and training accuracy $\gamma$.
Output: A set of rules $RS$.

    $RS = \emptyset$
    for all $c \in V_d - \{\top\}$ do
        $\mathcal{P} = \overline{A}\,\overline{G}\,\overline{K}^*_c$
        $\mathcal{N} = \underline{A}\,\underline{G}\,\underline{P}^*_{\sim\{c\}} \cap \overline{A}\,\underline{G}\,\underline{K}^*_{\sim\{c\}}$
        $RS = RS \cup \text{LearnRules}(\mathcal{P}, \mathcal{N}, A, d, c, \gamma)$
    end for
    return RS
```

Algorithm 8.4: Learning in the approximate ensemble approach

```
EnsemblePredictor:
Input: A G-elementary set $[x]_G$, a set of rules $RS$, a partial order $\langle V_d, \succcurlyeq \rangle$, and $\theta$.
Output: A set $P$ of class predictions for $[x]_G$.

 1: $RS' = \{(\alpha \to \beta) \in RS \mid [x]_G \subseteq [\![\alpha]\!]_{\mathcal{A}}\}$
 2: $Q = \{c \in V_d \mid (\alpha \to \langle d, c \rangle) \in RS'\}$
 3: for all $c \in V_d$ do
 4:     $RS'_c = \{(\alpha \to \beta) \in RS' \mid \beta = \langle d, c \rangle\}$
 5:     $SA(c) = \sum_{r \in RS'_c} NA(r)$
 6:     $SC(c) = \sum_{r \in RS'_c} NC(r)$
 7:     $WB(c) = \sum_{r \in RS'_c} WB(r)$
 8:     $Z = \{e \in (\sim\{c\} \cap Q) \mid$ there is no $f \in (\sim\{c\} \cap Q)$ such that $e \succ f\}$
 9:     $SN(c) = \sum_{r \in Z} 0.5 \cdot (SU(r) + SL(r))$
10: end for
11: $(P, PGain, EstWB) = SelectClasses(\top, A, \langle V_d, \succcurlyeq \rangle, \theta)$
12: return $(P)$
```

Algorithm 8.5: Prediction in the approximate ensemble approach

8.4 The bottom-up pruning method

The application of rules in the ensemble method is not very efficient. The rules created for the most specific classes will typically cover few objects, and their predictions will not be very accurate. Hence, they will contribute very little to the overall performance of the classifier, and it may actually be better to remove them all together.

Moreover, the objects that belong to a class c are not only covered by rules of c, but also by the rules of any superclass of c. If these objects can be predicted correctly to c, it is not necessary that the rules of the superclasses cover them as well. The rules of c would be sufficient. Thus the rules of the superclasses are redundant and could possibly be removed.

However, this redundancy is important in the ensemble approach since it allows the prediction algorithm to estimate votes and determine which classes should be predicted. The prediction algorithm would not work without them. Still, we may not need the prediction algorithm if we can determine which classes that can be predicted accurately and only learn rules for them. So, this redundancy may be removed.

This is the idea behind the method that will be presented in this section. The most detailed classes that can be predicted accurately are identified, and the classifier is built from the rules of these classes. The trade-off between detail level and precision is thus determined directly by the learning algorithm in this approach.

The classes, which can be predicted accurately, can be found efficiently by examining the DAG in a bottom-up fashion. We start with the leaf classes and move upwards. For each class, we create a set of rules and determine if the rules give accurate predictions. If the rules are of high quality, they are retained, and the objects covered by them are removed (such that they are not considered when rules are learned for the superclasses). If the rules are inaccurate, they are pruned, and the objects of the class are passed to the immediate superclass(es). The procedure is then repeated for the superclasses.

8.4.1 Learning

The details of the learning algorithm are presented in Algorithm 8.6. The algorithm is executed by a call to the procedure **LearnBottomUp**, which calls the recursive procedure **RecLearnBottomUp** for each class on the top level immediately below the root. **RecLearnBottomUp** performs the main task. It traverses the DAG depth-first and considers each class in a postfix order so that rules are learned first for the subclasses and then for the class itself. As there may be several paths to a class c, the algorithm checks initially if the class has been visited already and attempts only to create rules (and visit the subclasses), if the class has not been visited before.

The algorithm attempts to create rules for each class except for the root[8]. This done in two different ways depending on whether a class occurs at top level imme-

[8]The root itself provides no information about an object since all objects belong to the root. So, if the classifier predicted the root for an object, it would just imply that the class of the object was unknown.

LearnBottomUp:
Input: A rooted DAG-decision system $\mathcal{A} = \langle U, A, d, \succcurlyeq \rangle$ with root $\top$, training accuracy γ, split ratio ρ, pruning accuracy δ, and pruning support σ.
Output: A set of rules RS.

```
1: Clear rule set $RS$ $\{RS = \emptyset\}$
2: for all $c \in Sub(\top)$ do {For all immediate subclasses of $\top$}
3:    $(RS, R) = \text{RecLearnBottomUp}(c, RS, \mathcal{A}, \gamma, \rho, \delta, \sigma)$
4: end for
5: return RS
```

RecLearnBottomUp:
Input: a class c, a set of rules RS a rooted DAG-decision system $\mathcal{A} = \langle U, A, d, \succcurlyeq \rangle$, training accuracy γ, split ratio ρ, pruning accuracy δ, and pruning support σ.
Output: A set of rules RS and a set of uncovered objects R.

```
 1: if class $c$ has not been visited then
 2:    Mark $c$ as visited
 3:    $R = \emptyset$
 4:    for all $e \in Sub(c)$ do {For all immediate subclasses of $c$}
 5:       $(RS, R') = \text{RecLearnBottomUp}(e, RS, \mathcal{A}, \gamma, \rho, \delta, \sigma)$
 6:       $R = R \cup R'$
 7:    end for
 8:    $R_c = \{x \in U \mid d(x) = c \text{ and } x \notin Cov_{\mathcal{A}}(RS_c^{\prec})\}$
 9:    $\mathcal{P} = \overline{G}\,(R_c \cup R)$ and $\mathcal{N} = \underline{G}\,\underline{K}^*_{\sim\{c\}}$
10:    if $c \in Sub(\top)$ and top level pruning is off then
11:       $RS = RS \cup \text{LearnRules}\,(\mathcal{P}, \mathcal{N}, A, d, c, \gamma)$
12:       $R = \emptyset$
13:    else
14:       $(\mathcal{P}_t, \mathcal{P}_v) = \text{SplitData}(\mathcal{P}, \rho)$ and $(\mathcal{N}_t, \mathcal{N}_v) = \text{SplitData}(\mathcal{N}, \rho)$
15:       $RS_1 = \text{LearnRules}\,(\mathcal{P}_t, \mathcal{N}_t, A, d, c, \gamma)$
16:       $RS_2 = \text{PruneRules}(RS_1, \mathcal{P}_v, \mathcal{N}_v, \delta, \sigma)$
17:       $RS = RS \cup RS_2$
18:       $R = \{x \in \mathcal{P} \mid x \notin Cov_{\mathcal{A}}(RS_2)\}$
19:    end if
20: else
21:    $R = \{x \in \overline{K}^*_c \mid x \notin Cov_{\mathcal{A}}(RS_c^{\prec})\}$
22: end if
23: return (RS,R)
```

Algorithm 8.6: *Learning in the bottom-up approach*

diately below the root or at a more detailed level. In both cases, the objects are divided into a positive set and a negative set, and rules are learned with the subroutine **LearnRules**. However, the pruning algorithm **PruneRules** is not always applied to rules that are created on the top level. The reason is that the classes at this level are the most general classes that may be predicted. So the rules created for them are the most accurate that can be obtained. Moreover, the objects that are not covered at a more detailed level, must at least be covered at this level. Otherwise, they will not be covered by any rule, and the classifier will not make any predictions for them. Rules are therefore not pruned at the top level by default. The rules created for the detailed classes, on the other hand, may be improved. Hence, these rules are pruned. Nevertheless, some rules at the top level may be of poor quality such that a better performance may actually be obtained by removing them. The algorithm is therefore equipped with an option that allows pruning at the top level as well.

The positive set and the negative set in this algorithm are slightly different from those in the ensemble method as the A-approximations are not applied. The positive set $\mathcal{P}$ is a subset of $\overline{G}\,\overline{K}^*_c$ and the negative set $\mathcal{N}$ is equal to $\underline{G}\,\underline{K}^*_{\sim\{c\}}$. This means that the discernibility of the objects is not considered with regard to the attributes when these sets are computed. Note that this is not a problem[9] since this discernibility is also considered indirectly by algorithm **LearnRules**, which creates the rules for a class It is possible to use the same A-approximations as in the ensemble method. However, the algorithm seems to work better without them since no voting system is employed with it (We have actually tried with A-approximations as well). Moreover, this simplifies the algorithm since the computation of A-approximations is complicated by splitting in line 14.

The positive set $\mathcal{P}$ is only a subset of $\overline{G}\,\overline{K}^*_c$ since the algorithm should not learn new rules for the objects that have already been covered at the subclasses. Hence, $\mathcal{P}$ contains only objects in $\overline{G}\,\overline{K}^*_c$ that have not been covered. This set can be computed efficiently as the algorithm moves through the DAG since $\overline{K}^*_c$ can be decomposed as follows:

$$\overline{K}^*_c = \{x \in U \mid c \succcurlyeq d(x)\} = \bigcup_{c \succcurlyeq d} X_d$$

where $X_d = \{x \in U \mid d(x) = c\}$. The computation is done at several points in the algorithm. The objects that have not been covered at the subclasses are collected and added to R in line 6. The objects that belong to the class are found and stored in R_c in line 8. In this case, it is possible that some of these objects are covered by rules, which have been created for the subclasses. These objects should be not covered by new rules and must be removed from R_c. This is achieved with the requirement that the objects in R_c must not be in $Cov_{\mathcal{A}}(RS^{\prec}_c)$. $RS^{\prec}_c$ denotes the rules in RS that have

[9]A similar approach is usually taken in machine learning when flat classifiers with multiple classes are created. In this case, the positive and the negative sets for a class c are created by assigning the objects labeled with c to the positive set and the rest to the negative set (see e.g.,[59, 1]). Hence, the discernibility of the objects are not considered with regard to the attributes when the positive and negative sets are computed.

been learned at the subclasses,

$$RS_c^{\prec} = \{(\alpha \rightarrow \langle d, e\rangle) \in RS \mid c \succ e\}$$

and $Cov_{\mathcal{A}}(RS)$ is the set of objects that are covered by the rules in RS.

$$Cov_{\mathcal{A}}(RS) = \{x \in U \mid x \in [\![\alpha]\!]_{\mathcal{A}} \text{ and } (\alpha \rightarrow \beta) \in RS\}$$

Hence, $Cov_{\mathcal{A}}(RS_c^{\prec})$ contains the objects that are covered by the subclass rules.

The positive set is computed from R and R_c after these have been determined. This is done in line 9. The algorithm will then learn rules for the class and prune these rules, and we end up with a set of rules RS_2 that have been accepted for the class. The objects in $\mathcal{P}$ that are not covered by the accepted rules in RS_2 must be passed up to the superclasses such that another attempt (to learn rules for these objects) can be made at these classes. R is therefore recomputed in line 18 such that it contains the objects in $\mathcal{P}$ that are not covered by the rules in RS_2. Unfortunately, this transfer of objects is complicated by the DAG since a class may be visited again if it has several parents. The contents of R is not stored since it would require too much memory to store this set for every class in DAG. The contents of R is therefore lost when algorithm moves upwards and must be recomputed if a class is revisited. This computation is done in line 21[10].

8.4.2 Pruning

The pruning subsystem is responsible for removing rules that cannot be predicted accurately. It can be designed in many different ways. Some of these choices are:

- **Single rules vs. full classes**: The pruning can be made on two different levels – either on the class level or on the rule level. In the first case, we consider all rules that have been learned for a class and estimate how well they predict the class. If their performance is unsatisfactory, all of them are pruned. In the second case, each rule is tested separately and pruned if its performance is not good enough.

  The latter option has an advantage over the former since the pruning is more fine-meshed in this case. The learning task may not have the same degree of difficulty for all objects of a class c. For example, it may be easier to learn accurate rules for some objects than for the rest. Accurate rules may thus be made for the easy objects while the rest of objects may be passed to a more general superclass and covered by rules created for these classes. With the former option, this is not possible. All of the objects must be covered either at c or at a superclass. So the classifier will either give more incorrect predictions

[10]The computations in lines 8 and 21 may seem expensive. In our implementation we do not actually match each object against the rules in $RS_c^{\prec}$. Instead we label each object with the class(es) of the rule(s) that cover(s) it, and only check if c is general than these labels when R_c and R are computed.

SplitData:
Input: A set X and splitting ratio ρ.
Output: A set T for training and a set V for validation

```
1: Compute the quotient set $X' = X/IND(G)$
2: Split $X'$ into $T'$ and $V'$ at random such that $|T'| = \rho \cdot |X'|$ and $|V'| = (1-\rho) \cdot |X'|$
3: $T = \{x \in X \mid [x]_G \in T'\}$ and $V = \{x \in X \mid [x]_G \in V'\}$
4: return $(T, V)$
```

Algorithm 8.7: Splitting of data

or lose the details of objects that could be predicted c. Note, however, that this may be the only option if another learning approach such as discriminant analysis or support vector machines is applied.

- **Validation sample**: The algorithm needs a validation sample in order to estimate the performance of the rules. The training sample can be used for this purpose. However, this may lead to overfitting as a rule that fits the training data perfectly may have a different performance on another data set. The estimated performance may thus be overly optimistic such that a rule may not be pruned even though it should.

  An alternative is to split the original training data into a training sample and a validation sample such that rules are learned from the training sample and pruned on the validation sample. Such a strategy is often used in machine learning to avoid overfitting. Unfortunately, this leaves less data for training which may be a problem especially for the most specific classes where the available data is already quite scarce.

- **Pruning criterion**: The pruning algorithm needs a criterion to determine whether a rule should be accepted or not. One possible criterion is to allow the user to specify the minimal acceptable accuracy and prune if the performance is below this value. Another is to use the rules that are learned for classes immediately below the root as a yardstick. These classes are most general class that may be predicted and their rules should thus have the best performance. The rules that are learned for some class c can then be compared to the rules of these classes, and if the performance is worst the rules of c can be pruned.

In the approach that is presented here we have chosen to prune each rule independently. Moreover, the objects are divided into a training sample and a validation sample such that overfitting is avoided, and a rule is pruned if its accuracy is below the pruning accuracy δ, which is specified by the user. A rule is also removed if its support is below the pruning support σ.

PruneRules:
Input: A sets of rules RS, a positive set $\mathcal{P}_v$, a negative set $\mathcal{N}_v$, pruning accuracy δ, and pruning support σ.
Output: A pruned rule set RS.

```
1: for all ($\alpha \leftarrow \beta \in RS$ do
2:    if $Accuracy(\alpha, \mathcal{P}_v, \mathcal{N}_v) < \delta$ and $Support(\alpha, \mathcal{P}_v) < \sigma$ then
3:       $RS = RS - \{\alpha \leftarrow \beta\}$   {The rule is pruned}
4:    end if
5: end for
6: return $RS$
```

Algorithm 8.8: Pruning of rules

The learning and pruning of rules are performed in lines 14-16 in Algorithm 8.6. The data in the positive set $\mathcal{P}$ and the negative set $\mathcal{N}$ are initially divided into training sets ($\mathcal{P}_t$,$\mathcal{N}_t$) and validation sets ($\mathcal{P}_v$,$\mathcal{N}_v$). This is done by the procedure **SplitData**, which is shown in Algorithm 8.7. This procedure splits a set X in two according to the partition induced by the objects (i.e., the genes) in the original DAG-decision system. All objects of a G-elementary set (i.e., a gene) are therefore put either in the training set or in the validation set. This is necessary as the objects that belong to the same G-elementary set (i.e., same gene) should not occur both in the training set and the validation set. Otherwise, the estimated accuracy on the validation set would be too optimistic, and rules that should be pruned, might be retained. How the G-elementary sets are divided on the training set and validation set is controlled by the splitting ratio s. It is typically set to 2/3 so that 2/3 of the G-elementary sets end up in the training set and 1/3 in the validation set.

After the algorithm has divided the data into a training and a validation sample, it learns rules from the training sample (line 15) and prunes these rules on the validation sample (line 16). The pruning algorithm is shown in Algorithm 8.8. This procedure examines each rule in RS_c and tests if a rule should be deleted. This situation occurs if the accuracy of the rule is below the pruning accuracy δ or the support is below the pruning support σ. The accuracy $Accuracy(\alpha, \mathcal{P}, \mathcal{N})$ and support $Support(\alpha, \mathcal{P}_v)$ are defined as follows:

$$\begin{aligned} Accuracy(\alpha, \mathcal{P}, \mathcal{N}) &= |[\![\alpha]\!]_{\langle \mathcal{P}, A\rangle}|/(|[\![\alpha]\!]_{\langle \mathcal{P}, A\rangle}| + |[\![\alpha]\!]_{\langle \mathcal{N}, A\rangle}|) \\ Support(\alpha, \mathcal{P}) &= |[\![\alpha]\!]_{\langle \mathcal{P}, A\rangle}| \end{aligned}$$

8.4.3 Prediction

Just as in the ensemble method, predictions are made for G-elementary sets (Remember that such a set corresponds to an object, i.e., a gene, in the original untransformed DAG-decision system). However, no voting system is applied with this algorithm. When predictions are made for a G-elementary set (i.e., gene), the rules that cover this set are identified and the classes of these rules are simply considered as the predictions.

8.5 Search algorithms

The methods, which were introduced in the last two sections, learn the rules of a class in the same manner. A positive set $\mathcal{P}$ and a negative set $\mathcal{N}$ are first determined for the class. A set of rules is then learned by a rule learning algorithm called **LearnRules**. The details of this algorithm were not discussed. So these details will be provided here.

There are actually many different search algorithms that can be used for finding rules. Here, two different search algorithms, which are used in our experiments, will be presented. Both search through a hypothesis space, which consists of conjunctions of descriptors. However, the search is conducted in different directions. One searches the hypothesis space in a top-down fashion. The other searches the space from the bottom and up to the top.

8.5.1 Top-down search

The top-down algorithm is displayed in Algorithm 8.10. It is a so-called covering or separate-and-conquer algorithm [59]. This means that it searches for one rule at the time. When it finds a rule that *covers* some objects of the positive set with a certain accuracy (γ), it *separates* these objects from the rest and *conquers* the remaining objects by repeatedly learning rules until all objects are covered.

The separate-and-conquer task is performed in the outer-while loop of Algorithm 8.10 (line 2). The loop terminates when $\mathcal{P}$ is empty. When an antecedent has been found, the objects covered by it are removed from $\mathcal{P}$ (line 18), and it is turned into a rule and added to the rule set RS (line 19). For convenience, an antecedent is represented as a set (I and I_{best}). The function $Ant(I)$ turns the set into a conjunction when a rule is created

$$Ant(I) = \bigwedge_{\langle a,v\rangle \in I} \langle a, v\rangle$$

The inner while-loop (line 5) conducts a hill-climbing search for the antecedent. Initially, the antecedent set I and the best antecedent set I_{best} are empty, and the set B contains all descriptors that may be added to I. In each iteration, the descriptor that has the highest score is added to I. The loop terminates when I has sufficient accuracy or no more attribute-value pairs may be added to I.

The accuracy $Acc(I, \mathcal{P}, \mathcal{N})$ and the score $Score(I, \mathcal{P}, \mathcal{N})$ are defined as

$$\begin{aligned} Acc(I, \mathcal{P}, \mathcal{N}) &= Accuracy(Ant(I), \mathcal{P}, \mathcal{N}) \\ Score(I, \mathcal{P}, \mathcal{N}) &= Support(Ant(I), \mathcal{P}) \cdot Accuracy(Ant(I), \mathcal{P}, \mathcal{N}) \end{aligned}$$

```
LearnRulesTopDown:
Input: Positive set $\mathcal{P}$, negative set $\mathcal{N}$, conditional attributes $A$, decision attribute
  $d$, decision class $c$, and training accuracy $\gamma$ $(0 < \gamma \leq 1)$.
Output: A set of rules $RS$

1: $RS = \emptyset$
2: while $\mathcal{P} \neq \emptyset$ do
3:    $I = \emptyset$ and $I_{best} = \emptyset$
4:    $B = \{\langle a, v\rangle \mid a \in A$ and $v \in V_a\}$
5:    while $Acc(I, \mathcal{P}, \mathcal{N}) < \gamma$ and $B \neq \emptyset$ do
6:       select $\langle a', v'\rangle \in B$ with the highest $Score((I \cup \{\langle a', v'\rangle\}), \mathcal{P}, \mathcal{N})$
7:       $I = I \cup \{\langle a', v'\rangle\}$
8:       $B = B - \{\langle a', v\rangle \mid v \in V_{a'}\}$
9:       if $Acc(I, \mathcal{P}, \mathcal{N}) > Acc(I_{best}, \mathcal{P}, \mathcal{N})$ or $I_{best} = \emptyset$ then
10:         $I_{best} = I$
11:      end if
12:   end while
13:   for each $\langle a, v\rangle \in I_{best}$ (in the order that they were added to $I$) do
14:      if $Acc((I_{best} - \langle a, v\rangle), \mathcal{P}, \mathcal{N}) \geq \gamma$ then
15:         $I_{best} = I_{best} - \{\langle a, v\rangle\}$
16:      end if
17:   end for
18:   $\mathcal{P} = \mathcal{P} - [\![Ant(I_{best})]\!]_{\langle \mathcal{P}, A\rangle}$
19:   $RS = RS \cup \{Ant(I_{best}) \rightarrow \langle d, c\rangle\}$
20: end while
21: return RS
```

Algorithm 8.9: Top-down search for rules of a class

Note that it is possible to use just the accuracy as score when descriptors are selected. However, when the support is multiplied with the accuracy, the algorithm is forced to consider both measures and not only the accuracy.

The best antecedent is maintained in I_{best} and is mainly used when the inner-while loop terminates by the second condition. In this case, the specified accuracy cannot be obtained and I_{best} contains the most general antecedent with the best accuracy. Note that I_{best} will be equal to I if the inner loop terminates by the first condition.

The inner while-loop performs a greedy search that may add redundant conditions to an antecedent. The antecedent set I_{best} is therefore examined and redundant conditions are removed before a rule is created. This is done in the for-loop at line 13. The descriptors are processed in the order that they were added to the antecedent set. A descriptor is deleted from I_{best} if the accuracy without it is above training accuracy.

Note that possible rules are found only if the training accuracy γ is 1. However,

LearnRulesBottomUp:
Input: Positive set $\mathcal{P}$, negative set $\mathcal{N}$, conditional attributes A, decision attribute d, decision class c, and training accuracy γ $(0 < \gamma \leq 1)$.
Output: A set of rules RS.

```
1: $R = \{Inf_A(x) \mid x \in \mathcal{P}\}$
2: while two antecedents in $R$ can be merged into $I$ and $Acc(I, \mathcal{P}, \mathcal{N}) \geq \gamma$ do
3:    select the two most similar $I_1, I_2 \in R$, i.e., those with the least $dist(I_1, I_2)$
4:    create $I = merge(I_1, I_2)$
5:    remove $I_1, I_2$ from $R$ and add $I$ to $R$
6: end while
7: $RS = \emptyset$
8: for each $I \in R$ do
9:    for each $\langle a, v \rangle \in I$ do
10:      if $Acc((I - \langle a, v \rangle), \mathcal{P}, \mathcal{N}) \geq \gamma$ then
11:         $I = I - \{\langle a, v \rangle\}$
12:      end if
13:   end for
14:   $RS = RS \cup \{Ant(I) \rightarrow \langle d, c \rangle\}$
15: end for
16: return RS
```

Algorithm 8.10: Bottom-up search for rules of a class

sometimes it is possible to obtain better results if this criterion is relaxed. Therefore, the algorithm allows the user to specify γ manually.

8.5.2 Bottom-up search

The bottom-up search algorithm is shown in Algorithm 8.10. Initially, a most specific antecedent set is created for each object in the positive set where the most specific antecedent of an object is essentially its information vector. The algorithm tries then to merge the two most similar antecedent sets into a more general antecedent set by dropping dissimilar descriptors. This merge operation can be described with the following function.

$$merge(I_1, I_2) = \{\langle a, v \rangle \mid \langle a, v \rangle \in I_1, \langle a, v \rangle \in I_2\}$$

The similarity of the antecedent sets is measured by

$$dist(I_1, I_2) = |\{a \in A \mid \langle a, v_1 \rangle \in I_1, \langle a, v_2 \rangle \in I_2, \text{ and } v_1 \neq v_2\}|$$

The generalization process is repeated as long as there are some antecedent sets that may be merged and the resulting antecedent set I has an accuracy above the training accuracy γ.

The antecedent may have redundant descriptors. So after the generalization process has terminated, each antecedent set is examined and redundant descriptors are removed. This is done in almost same manner as in the top-down search. However, the algorithm does not add descriptors to the antecedent such that the descriptors are just processed in the order that they appear in the antecedent set.

Evaluation Methods for DAG learning

9

Some parts of this chapter have been published in [115].

9.1 Introduction

The performance of supervised learning methods is usually measured by the *accuracy* or the *area under the ROC curve* (AUC) (see e.g., Chapter 5). Some authors also consider the *error rate*, which is simply $1 - accuracy$. All of these measures assume, however, that each object has a unique decision class and that only one prediction is made for each object. Moreover, a prediction is either correct (if it is identical to the decision class of the object) or incorrect (if it is different from the decision class). However, a prediction cannot be partially correct.

These assumptions do not hold in our case. As we have discussed previously, a gene may be annotated with several decision classes, and several predictions can be made for each gene. A prediction need not be identical to a decision class, either. It may be above or below the decision class such that it matches the class only partially.

Thus, standard performance measures of supervised learning are not applicable. In this chapter, we introduce a set of measures for learning with DAG-decision system with multiple decision classes (as defined in Definition 7.12).

9.2 Measuring multiple annotations and predictions

We begin by considering the problem where an object has multiple decision classes and predictions and ignore the ontology for now. In this case, we have a decision system $\mathcal{A} = \langle U, A, D \rangle$ with multiple decisions per object where $D(x)$ is a set of classes annotated to object $x \in U$. Moreover, we have a classifier, which is to be evaluated and a set of predictions $\hat{D}(x)$ that are made by the classifier for each object $x \in U$. There are two ways that this classifier may fail:

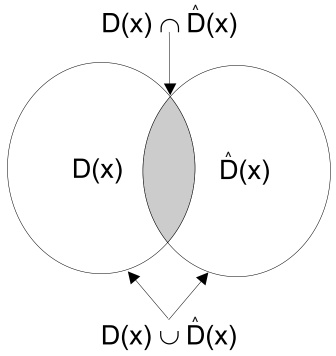


Figure 9.1: *Measure for similarity of annotations and predictions for object x.*

- It may not predict a class c that should be predicted (i.e., $c \in D(x)$ and $c \notin \hat{D}(x)$).
- It may predict a class c that should not be predicted (i.e., $c \notin D(x)$ and $c \in \hat{D}(x)$).

These errors can be both assessed with a metric such as

$$\frac{|D(x) \cap \hat{D}(x)|}{|D(x) \cup \hat{D}(x)|}$$

which measures how many classes that $D(x)$ and $\hat{D}(x)$ have in common for object x (as illustrated in Figure 9.1). It will be 1 if $D(x) = \hat{D}(x)$ and 0 if $D(x)$ and $\hat{D}(x)$ are disjoint ($D(x) \cap \hat{D}(x) = \emptyset$). The average similarity over all objects can then be found with

$$\frac{1}{|U|} \sum_{x \in U} \frac{|D(x) \cap \hat{D}(x)|}{|D(x) \cup \hat{D}(x)|}$$

This measure is useful for comparing the performance of different classifiers such that the best classifier can be identified. However, it is not very helpful in identifying in what way the classifier fails when it achieves less than a full score. For example, if we want to improve the performance of the classifier, we need to know what kind of mistakes it makes — does it make too few predictions such that some annotations are not predicted or does it make too many such that some predictions do not match any annotations? Unfortunately, both kinds of errors are measured in the same way by this measure. So, there is no way to tell.

One way to solve this problem is to use two measures – one for each kind of error. We assess the ratio RA of annotations that are predicted, and the ratio RP of predictions that are correct, i.e., those that correspond to annotations.

Definition 9.1 (Global Recall/Precision). Let $\mathcal{A} = \langle U, A, D \rangle$ be a decision system with multiple decision classes such that an object $x \in U$ is labeled with set of classes $D(x)$. Assume also that $\hat{D}(x)$ is a set of predictions made for object x by a classifier.

- **Recall**: $RA = \frac{\sum_{x \in U} |MA(x)|}{\sum_{x \in U} |D(x)|}$ where $MA(x) = D(x) \cap \hat{D}(x)$
- **Precision**: $RP = \frac{\sum_{x \in U} |MP(x)|}{\sum_{x \in U} |\hat{D}(x)|}$ where $MP(x) = D(x) \cap \hat{D}(x)$ ■

These metrics were believed to be novel when they were introduced in [115]. However, similar scores have been used for a long time in Information Retrieval (see e.g., [89, Chapter 8] and [5, Chapter 3]) where they are called *recall* and *precision*. We will therefore adopt these names here.

Note that there is an alternate way to define these measures. We may consider the ratio of annotations on an object-wise basis and then compute the average ratio over all objects (RA'). Similarly, we find the average ratio of the predictions that are correct for each object (RP').

Definition 9.2 (Average object-wise recall/precision).

- **Recall**: $RA' = \frac{1}{|U|} \sum_{x \in U} \frac{|MA(x)|}{|D(x)|}$
- **Precision**: $RP' = \frac{1}{|U|} \sum_{x \in U} \frac{|MP(x)|}{|\hat{D}(x)|}$ ■

The measures in Definition 9.1 may be viewed as weighted averages with respect to the objects where the weights depend on the number of annotations (for RA) or predictions (for RP) that an object has. Objects with many annotations/predictions will affect the measures more than objects with few annotations/predictions. For example, for recall we have:

$$RA = \frac{\sum_{x \in U} |MA(x)|}{\sum_{x \in U} |D(x)|} = \sum_{x \in U} \left(\frac{|MA(x)|}{|D(x)|} \cdot \frac{|D(x)|}{\sum_{x \in U} |D(x)|} \right) = \sum_{x \in U} \frac{|MA(x)|}{|D(x)|} \cdot w_x$$

The measures given in Definition 9.2 weight the objects equally. However, this means that one missing annotation will affect RA' less if an object has 10 annotations than if it has only 2.

9.3 Measuring the DAG

We will now consider measures for evaluating a classifier trained on a DAG-decision system with multiple decisions $\mathcal{A} = \langle U, A, D, \succcurlyeq \rangle$. As before, we will assume that the classifier makes a set of predictions $\hat{D}(x)$ for each object $x \in U$.

Recall and precision may also be used in this case. However, they consider an annotation a and a prediction p to match only if a and p are equal. This is clearly a too strict requirement for the DAG as a and p may match partially if they are related. The set of matched annotations $MA(x)$ and the set of matched predictions $MP(x)$ are therefore redefined so that an annotation is considered matched if there is a related prediction, and a prediction is matched if there is a related annotation:

$$\begin{aligned} MA(x) &= \{a \in D(x) \mid p \in \hat{D}(x) \text{ and } a \approx p\} \\ MP(x) &= \{p \in \hat{D}(x) \mid a \in D(x) \text{ and } p \approx a\} \end{aligned}$$

The definitions of recall and precision as given in Definitions 9.1 and 9.2 can then be applied on these sets instead of $MA(x) = MP(x) = D(x) \cap \hat{D}(x)$.

These measures estimate the number of the annotations and number of the predictions that are partially matched. However, they do not consider the loss of details that may occur when a prediction is more general than an annotation. For example, if an object is labeled with *oxidative stress response* and the superclass *stress response* is predicted (see Figure 9.2), some of the original detail level has been lost. The prediction does not tell us which subclass of *stress response* that object belongs to. A classifier that predicts *oxidative stress response* is therefore better than a classifier that predicts *stress response*, and we would select the first classifier if we had a choice. However, it is not possible to distinguish between such classifiers if only recall and precision are used. Hence, the loss of details should be assessed so that such classifiers may be told apart.

In order to quantify this loss, we consider the depth of a class which is equal to the length of the path from the class to the root (i.e., the number of edges from the class to the root). There may be more than one path from a class to a the root in a DAG so that a class may occur at several depths. The depth of a class is therefore defined with respect to a particular path.

Definition 9.3 (Depth). Let t be a path from $c \in V_d$ to the root $\top$. The depth of a class c with respect to path t, denoted $Depth_t(c)$, is $||t||$. ■

The loss associated with a prediction p that is more general than annotation a, can be measured as the depth of p relative to the depth of a.

Definition 9.4 (Relative depth). Given an annotation a and a prediction p with the associated paths t_a and t_p, the relative depth of a and p with respect to t_a and t_p is:

$$RDepth_{t_a,t_p}(a,p) = \begin{cases} \frac{Depth_{t_p}(p)}{Depth_{t_a}(a)} & \text{if } t_a \sqsubseteq t_p \text{ or } t_p \sqsubseteq t_a \\ 0 & \text{otherwise} \end{cases}$$

where $s \sqsubseteq t$ denotes that s is a subpath of t (see Definition 7.9 in Section 7.3). ■

This measure is illustrated in Figure 9.2. It will be 1 if a and p have the same depth; less than 1 if p is a superclass of a; and 0 if the a and p are unrelated. It also

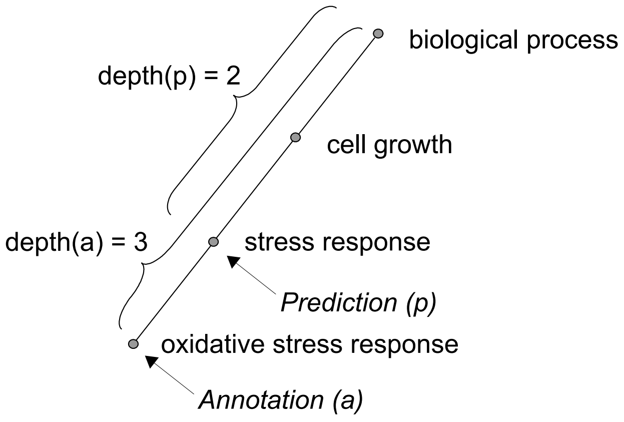


Figure 9.2: *Depth and relative depth.* Assume that an object is labeled $a = \mathit{oxidative\ stress\ response}$ and a prediction is made for this object to $p = \mathit{stress\ response}$. In this case, there are two edges from p to the root such that p has depth $Depth(p) = 2$. Similarly, a has depth 3. The relative depth is $RDepth(a, p) = \frac{Depth(p)}{Depth(a)} = \frac{2}{3}$.

measures the gain in detail level that is obtained when a prediction is more specific than an annotation. In this case, $RDepth_{t_a,t_p}(a, p) > 1$.

Note that one of the paths t_a and t_p in $RDepth_{t_a,t_p}(a, p)$ must be a subpath of the other. Otherwise, the paths are not comparable. For example, if $p \succcurlyeq a$ and a has several immediate superclasses, there will be some paths from a to the root that include p and some that do not. The paths that do not contain p cannot be compared against the paths of p. So only paths that contain a path from p to the root as subpath should be considered.

$RDepth_{t_a,t_p}(a, p)$ depends on the paths t_a and t_p, and there may be several different relative depths associated with a and p. We need a single measure on how well p reassembles a. To this end, the maximal relative depth is used.

Definition 9.5 (Maximal relative depth). Let a be an annotation and p a prediction. The maximal relative depth of a and p is defined as

$$mrd(a, p) = \max_{\substack{t_a \in Paths(a,\top), \\ t_p \in Paths(p,\top)}} RDepth_{t_a,t_p}(a, p)$$

■

$mrd(a, p)$ measures not only the loss (when $p \succcurlyeq a$), but also the gain (when $a \succcurlyeq p$). However, when we evaluate the classifier, we are mainly interested in how well the classifier reproduces the actual annotations, and are less interested if some details are gained. The maximal relative depth $mrd(a, p)$ is therefore restricted such that no additional points are given if p is more detailed than a.

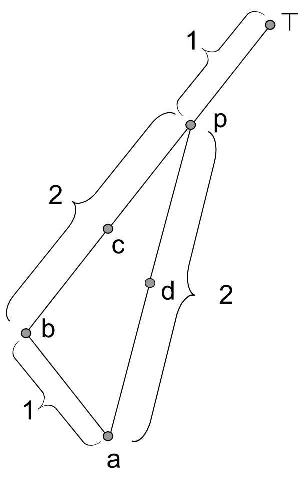


Figure 9.3: *Maximal relative depth.* There are two path from a to $\top$ and one path from p to the $\top$. $Depth_{\langle a,b,c,p,\top\rangle}(p) = 4$, $Depth_{\langle a,d,p,\top\rangle}(p) = 3$, and $Depth_{\langle p,\top\rangle}(p) = 1$ The maximal relative depth is $mrd(a,p) = (Depth_{\langle p,\top\rangle}(p)/Depth_{\langle a,d,p,\top\rangle}(p)) = 1/3$.

Definition 9.6 (Bounded maximal relative depth). Let a be an annotation and p a prediction. The bounded maximal relative depth of a and p is

$$bmrd(a,p) = \max(mrd(a,p), 1) \qquad \blacksquare$$

$bmrd(a,p)$ measures only the loss in a prediction with regard to a single annotation. In order to capture the loss in all annotations and objects, we introduce two measures that accompany recall and precision.

Definition 9.7 (Average recall/precision depth). Let $\mathcal{A} = \langle U, A, D, \succcurlyeq \rangle$ be a DAG-decision system with multiple decision classes, and let $\hat{D}(x)$ be a set of predictions made for object x by a classifier. Then the average recall and precision depths are:

- **Avg. recall depth**: $DA = \dfrac{\sum_{\substack{x\in U \\ a\in D(x)}} \max_{p\in\hat{D}(x)} bmrd(a,p)}{\sum_{x\in U} |MA(x)|}$
- **Avg. precision depth**: $DP = \dfrac{\sum_{\substack{x\in U \\ p\in\hat{D}(x)}} \max_{a\in D(x)} bmrd(a,p)}{\sum_{x\in U} |MP(x)|}$ $\blacksquare$

The recall depth DA is the average $bmrd$ of the best matching prediction for each matched annotation. This gives an indication of how well the annotations are reproduced. The precision depth DP gives the average $bmrd$ for each prediction and their best matching annotation, indicating how well each prediction matches.

Note that $bmrd(a, p)$ can be replaced with $mrd(a, p)$ in DA and DP if one wants to measure the gain as well as the loss in the predictions. However, reductions in the detail level may not be detected in this case. For example, a loss in the detail level of one prediction may be compensated by a similar gain of another prediction so that DA and DP may be approximately 1 or higher. Consequently, it may be better to use an unbounded version of DA and DP together with DA and DP if the gain is to be measured.

This also holds more generally. With many different sources of errors, it will usually be better to measure each source separately than measuring several with a single measure. It is possible to combine the recall RA' and the recall depth DA into a single measure:

$$\frac{1}{U}\sum_{x\in U}\frac{\sum_{a\in D(x)}\max_{p\in\hat{D}(x)} bmrd(a,p)}{|MA(x)|}$$

A similar measure can also be created with precision RP' and DP. One may even merge all of these measures into a single measure such as

$$\frac{1}{U}\sum_{x\in U}\frac{\sum_{a\in D(x)}\max_{p\in\hat{D}(x)} bmrd(a,p)+\sum_{p\in\hat{D}(x)}\max_{a\in D(x)} bmrd(a,p)}{|MA(x)|+|MP(x)|}$$

However, none of them are easily interpreted. So they are not very useful (except for the last one that could be used as single metric for comparing classifiers). We will therefore use the four measures recall (RA), recall depth (DA), precision (RP), and precision depth (DP) in the experiments in the next chapter.

Experiments and Results

10

A previous version of the ensemble method has been applied to the fibroblast data. The results obtained with that version were published in [115] and were quite similar to the ones that are presented in this chapter.

10.1 Introduction

This chapter reports on several experiments with the algorithms that were introduced in Chapter 8. The algorithms are first applied to a real life data set created with microarrays. We then examine the performance of the algorithms on a series of artificial data sets.

10.2 The fibroblast data

Our first experiment was on a data set created by Iyer et al. [76]. They studied the gene response in human fibroblast cells[1] with cDNA microarrays. In their experiments, growth factor serum was initially removed for 48 hours from a cell culture. This forced the cells into a quiescent state. Growth factor serum was then added, and samples were collected at 12 different time points (0 h, 15 min, 30 min, 1 h, 2 h, 4 h, 8 h, 12 h, 16 h, 20 h, and 24 h) and analyzed with cDNA microarrays. Each microarray contained about 8600 genes. Iyer at al. found that 517 of the genes showed substantial changes in their expression levels. These genes were clustered using hierarchical clustering, and 10 major groups were identified by inspecting the dendrogram and the heat map created by the clustering algorithm.

[1]Fibroblasts are connective tissue cells that take part in wound healing and are capable of differentiating into specialized cells such as cartilage, bone, fat, and muscle cells. In culture, they require growth factors for proliferation (i.e., multiplication/reproduction). Growth factors are usually provided by fetal bovine serum.

10.2.1 Data material

The data set[2] provided by Iyer et al. contained the 517 differentially expressed genes and their associated expression profiles. However, it did not contain any annotations for the genes. The genes were therefore annotated manually by using relevant information from the literature and molecular biology databases (see [90] for details). The annotation classes were taken from biological process subontology (rev. 1.1152 - 25-Aug-2000) of the GO Consortium. A total of 723 annotations were found for 323 of the genes in the data set. 203 of these genes were annotated with more than one class. In the following, the annotated data set is referred to as the fibroblast data.

The annotations referred to only 234 classes so that most of the classes in the process ontology were not used. In particular, no gene was annotated with any of the leaf classes in the ontology. Therefore a part of the ontology was selected and only this part was used in the experiment. The part consisted of the most specific classes with annotations and their superclasses. It had a total of 313 classes where 157 were leaf classes. 113 of the classes were associated with only one gene. As explained in Section 6.8, the inheritance rule was not applied in this experiment. Hence, the ontology was *not* transformed. The difference between **is-a**- and **part-of**-relationships was simply ignored.

The microarray measurements were real numbers. However, the rule learning algorithms assumed that attribute values were discrete. Thus, it was necessary to discretize the measurements after $\log_2$-transformation and normalization[3]. This was achieved with the template approach introduced by Hvidsten et al. [75].

This approach transforms real numbers in the expression profile of a gene into templates. A template is simply an attribute value that describes a pattern over several consecutive time points. Hvidsten et al. distinguish between three different templates: Increasing (*up*), decreasing (*down*), and constant (*const*). A simplified variant of the templates appeared in Table 8.1 where each attribute is constructed from only adjacent time points.

In this experiment, we created templates that stretched over 2 to 6 time points. Thus, attributes such as 0H-15min, 0H-30min, 0H-1H, 0H-2H, and 0H-4H were constructed. Templates were then assigned to each attribute and object as follows:

- An *up* template was created if the slope between the end points in the interval, which was defined by the attribute, was greater or equal to 0.03. If the interval stretched over 3 or more time points an additional requirement was set such that the slope between adjacent time points in the interval could not be below -0.02.

- A *down* template was constructed if the slope between the end points was less than or equal to -0.03. Moreover, if the interval stretched over 3 or more time

[2]This is available at **http://genome-www.stanford.edu/serum/**.

[3]Each expression profile was normalized as follows: Let X_i $(1 \leq i \leq 12)$ be $\log_2$-ratios for a gene over the 12 time points. A normalized ratio Y_i was then computed as: $Y_i = X_i / \sqrt{\sum_{i=1}^{12} X_i^2}$

Method	Parameter	Description	Used in algorithm
Bottom-up	Top level pruning	Pruning at classes immediately below the root (Yes/No)	8.6
	ρ	Split ratio	8.7
	δ	Pruning accuracy	8.8
	σ	Pruning support	8.8
Ensemble	θ	Threshold for adjusting the correction of the WB support	8.3
Top-down search	γ	Training accuracy	8.9
Bottom-up search	γ	Training accuracy	8.10

Table 10.1: *Overview of the parameters used in the learning algorithms.*

points, the slope between any adjacent time points in the interval could not be above 0.02.

- A *const* template was assigned if the absolute value of the slope between the endpoints was less than 0.03. If the interval stretched over 3 or more time points, the absolute value of the slope between any adjacent time points in the interval had to be below 0.02 as well.
- If none of the above conditions could be fulfilled, no template was assigned to the object for this attribute.

10.2.2 Results

We applied the bottom-up and ensemble methods to the data set and estimated their performance with 10-fold cross-validation. The results are reported in Table 10.2 and were quite similar for all of the methods (A summary of the parameters used by the methods is given in Table 10.1). Most of the annotations in the fibroblast data were predicted, and most of the predictions were correct. However, a lot of the original detail level was lost. The bottom-up method retained slightly more details than the ensemble method, but had a lower precision than the ensemble method. The performance of two versions of the ensemble method was very similar.

The algorithms were compared to two "flat" algorithms that ignored the DAG structure. These learned possible rules for each class in a similar fashion to the ensemble method. However, the positive sets and negative sets were defined as they would be defined for a rough set classifier that predicts several classes for each object. So when the algorithms learned rules for the a class c, the positive set $\mathcal{P}$ was $\overline{AG}X_c$ and the negative set $\mathcal{N}$ was $\underline{AG}(U - X_c)$. When rules were found for the complement $\mathcal{P}' = \overline{A}\underline{G}(U - X_c)$ and $\mathcal{N}' = \underline{A}\overline{G}X_c$.

The first approach[4] (described as "flat (voting)" in Table 10.2) learned rules for a

[4]A similar classifier could be obtained with a rough set system such as ROSETTA by handling each

Method	*Search*	γ	*Top lev. pru.*	*Match (Global) Recall (RA)*	*Match (Global) Precision (RP)*	*Depth Recall (DA)*	*Depth Precision (DP)*
Bottom-up							
	Top-down	0.8	***Yes***	**0.79**±0.02	**0.72**±0.02	**0.29**±0.01	**0.32**±0.01
	Bottom-up	0.8	***Yes***	**0.79**±0.02	**0.73**±0.03	**0.29**±0.01	**0.32**±0.01
	Top-down	0.8	***No***	**0.85**±0.02	**0.63**±0.01	**0.30**±0.01	**0.34**±0.01
	Bottom-up	0.8	***No***	**0.84**±0.02	**0.59**±0.02	**0.29**±0.01	**0.32**±0.01
Ensemble							
(Normal)	***Top-down***	0.9		**0.73**±0.02	**0.76**±0.02	**0.28**±0.01	**0.31**±0.01
	Bottom-up	0.9		**0.79**±0.01	**0.82**±0.01	**0.27**±0.01	**0.31**±0.01
	Top-down	0.8		**0.79**±0.02	**0.86**±0.02	**0.27**±0.01	**0.30**±0.01
	Bottom-up	0.8		**0.79**±0.02	**0.87**±0.01	**0.27**±0.01	**0.30**±0.01
Ensemble							
(Approx)	***Top-down***	0.9		**0.73**±0.03	**0.78**±0.02	**0.27**±0.01	**0.31**±0.01
	Bottom-up	0.9		**0.77**±0.01	**0.80**±0.02	**0.27**±0.01	**0.30**±0.01
	Top-down	0.8		**0.79**±0.02	**0.82**±0.02	**0.27**±0.01	**0.31**±0.01
	Bottom-up	0.8		**0.79**±0.02	**0.82**±0.02	**0.27**±0.00	**0.30**±0.01
Flat							
(Voting)	***Top-down***	0.8		**0.00**±0.00	**0.00**±0.00	**0.00**±0.00	**0.00**±0.00
	Bottom-up	0.8		**0.00**±0.00	**0.00**±0.00	**0.00**±0.00	**0.00**±0.00
Flat							
(No voting)	***Top-down***	0.8		**0.14**±0.01	**0.11**±0.01	**0.95**±0.01	**0.95**±0.01
	Bottom-up	0.8		**0.12**±0.01	**0.09**±0.01	**0.89**±0.03	**0.90**±0.02

Table 10.2: *Results on the fibroblast data.* For each measure, the average over the 10 cross-validation folds is shown in bold. The quantity after the ± sign is the standard error. The bottom-up algorithm was executed with $\rho = 0.66$, $\delta = 0.8$, and $\sigma = 1$. θ was set to 0.7 for the ensemble method.

class and its complement. For prediction, it used the voting procedure in Section 4.7. This procedure was applied to each class such that a class was predicted if it had a higher certainty than its complement.

The second approach (described as "flat (no voting)") learned only rules for a class, but not for the complement. It did not use any voting procedure. The predictions for an object consisted simply of the classes in the rules that covered the object.

Table 10.2 also presents the results obtained with these two algorithms. These results were very weak. The first algorithm did not make any predictions at all because the certainty of each class was always lower than the certainty of the complement. The second algorithm worked slightly better. Approx. 10% of the actual annotations

class as a separate prediction task. In this case, we would create a new training set for each class c so that an object labeled to another class than c would be labeled with "not c", and an object annotated to both c and another class would be labeled with just "c". The training set would then be given to ROSETTA.

were predicted, and most of the details of these annotations were retained. However, about 90% of the predictions made by the classifier were incorrect. Hence, the results obtained with the bottom-up and ensemble methods were obviously better. By trading off details, they were able to reproduce many more annotations with much higher precision. This demonstrated the need for taking the DAG-structure into account during learning.

The predictions made by our algorithms had lost a lot of the details of the annotation. Most of them were made to the classes immediately below the root. We believe that this was due to the nature of the fibroblast data. The variation in the data may not be sufficient to distinguish between the more detailed classes. Iyer et al. found only 10 major clusters in the data indicating that no more than 10 classes may be distinguished. So, it seems highly unlikely that all 157 leaf classes in our annotations may be discerned. The genes participating in different processes may simply be similarly expressed in the experiment of Iyer et al., and in order to distinguish between the more detailed classes, we would require more microarray experiments. Thus, the essential variations in the fibroblast data may be best captured by the general classes at the top of the ontology.

The fibroblast data have also been classified with Rosetta– originally, in Hvidsten et al. [75] and more recently in Lægreid et al. [90]. In both studies a subset of the classes in the ontology was selected, and the genes, which were annotated to more specific classes, were relabeled with the selected classes. In Hvidsten et al. the most specific classes, which had at least 10 genes annotated to either themselves or their subclasses, were selected. This resulted in a set of 16 classes. In Lægreid et al. 23 classes were selected manually using biological knowledge about the data.

The results obtained in these studies are summarized in Table 10.3. It should be mentioned that slightly different versions of the annotations were used in these studies, and that an intermediate version of these annotations was used in the experiments. There were also some differences in the definitions of the templates. This meant that only a rough comparison was possible. However, it appeared that our results had a higher precision, while more details were retained in the other studies. The precision in Hvidsten et al. was very low, and even in Lægreid et al. about 50% of the predictions were incorrect. Compared to our results, it seemed that too many details were kept in these experiments. So at least some of the selected classes in these studies should have been replaced by more general classes.

A question is obviously whether our algorithms predicted too general classes such that details were lost needlessly. This could have occurred if the algorithms had a tendency to predict too general classes. This issue is examined in the next section through controlled experiments with artificial data. The results that will be reported in that section showed that the bottom-up method retained the detail level quite well. The ensemble method could lose details, but this depended on the choice of θ. The experiments on the artificial data showed that the loss was insignificant when $\theta = 0.7$, which is the setting that was used with the classifiers in Table 10.2. Hence, it is not likely that details were lost because of these algorithms.

Still, the detail level of the predictions did not depend solely on the bottom-up or

Study	***No. of Classes***	***Match (Global)***		***Depth***	
		Recall (RA)	***Precision (RP)***	***Recall (DA)***	***Precision (DP)***
Hvidsten et al. [75]	16	0.67	0.33	0.71	0.71
Lægreid et al. [90]	23	0.84	0.49	0.69	0.69

Table 10.3: *Results obtained with* Rosetta *on the fibroblast data.* These figures are only rough estimates as Hvidsten et al. only reported sensitivity and specificity for each class[6]. Lægreid et al. reported recall and precision so that these numbers are exact. However, recall depth and precision depth were not reported in either study. These figures were estimated by comparing the original annotations to the "moved" annotations that were created by relabeling the genes with the selected classes.

ensemble method. The search algorithms, which were used for learning rules, could also have affected the detail level. For example, if a search algorithm created poor rules, they could have been pruned by the bottom-up approach such that objects were pushed upwards, and the detail level was reduced. Poor rules could also have led to inaccurate votes, which again could have caused the algorithm to predict too general classes. A related question is therefore whether more detail could have been obtained if another search algorithm had been used for learning rules for each class.

The two search algorithms that were used here were fairly simple. They attempted to find a minimal set of rules so that one rule at most was created for each object. This made them quite sensitive to noise. For example, when the algorithms built a rule r, they might have to choose between several attributes that had the same ability to discern between the objects in the positive and the negative set. Only one of these attributes would be selected since the algorithms tried to create rules with a minimal number of attributes. This meant that a new object, which had a distorted value for the chosen attribute (but was otherwise similar to the objects covered by r) would not be covered by r. Hence, no class would be predicted for this object. The classifier would consequently be sensitive to noise in the selected attribute. However, the object might have been covered if one of the other attributes had been selected instead since it is not likely that all of its attribute values had been changed by noise. So, if the algorithms made several rules – one for each of these attributes — one of the rules would probably cover the object, and the algorithms would be less sensitive to noise.

[6]Note that sensitivity and specificity are useful when a single class is considered, but they do not give a good impression of the performance for full classifiers (consisting of multiple classes). In particular, the specificity may be misleading as measure of the correctness. The reason is that for any given class, the positive set, containing the genes annotated to the class, is much smaller than the negative set, containing genes annotated only with other classes. Thus, the number of false positives may be equal or higher than the number of true positives even with a quite high specificity. This effect is further increased by the fact that multiple classes are predicted for each gene since the errors made individually for the classes add up in this case. So even with a seemingly high specificity, the number of false predictions may be much higher that the number of true predictions, resulting in a low RP.

Method	*Search*	γ	*Top lev. pru.*	*Match (Global)* Recall (RA)	*Precision (RP)*	*Depth* Recall (DA)	*Precision (DP)*
Bottom-up							
	Top-down	0.8	***Yes***	**0.81**±0.01	**0.56**±0.02	**0.36**±0.01	**0.39**±0.01
	Bottom-up	0.8	***Yes***	**0.80**±0.02	**0.54**±0.01	**0.34**±0.01	**0.37**±0.01
	Top-down	0.8	***No***	**0.84**±0.01	**0.49**±0.02	**0.35**±0.01	**0.38**±0.01
	Bottom-up	0.8	***No***	**0.83**±0.02	**0.50**±0.01	**0.35**±0.01	**0.37**±0.01
Ensemble (Normal)	***Top-down***	0.9		**0.56**±0.02	**0.58**±0.02	**0.32**±0.01	**0.38**±0.01
	Bottom-up	0.9		**0.63**±0.02	**0.69**±0.02	**0.33**±0.01	**0.37**±0.01
Ensemble (Approx)	***Top-down***	0.8		**0.53**±0.02	**0.54**±0.02	**0.34**±0.00	**0.39**±0.01
	Bottom-up	0.8		**0.50**±0.02	**0.53**±0.02	**0.34**±0.01	**0.41**±0.02

Table 10.4: *Results on the fibroblast data with relaxed settings.* The bottom-up method was run with $\rho = 0.66$, $\delta = 0.1$, and $\sigma = 1$. θ was set to 1 for the ensemble method.

Hvidsten et al. and Lægreid et al. used the genetic reduct algorithm described briefly in Section 5.2.3 and created object-wise reducts. This algorithm created several rules for each object and was most likely less sensitive to noise than the two search algorithms used in this study. This was quite evident when we compared the number of rules used in their approaches to the number of rules used in our algorithms. The ensemble method created the most rules in our approach. It produced 1855 rules when top-down search was used and 1679 rules when bottom-up search was applied. In Lægreid et al. a total of 18064 rules were created. So, the genetic reduct algorithm created 10 times as many rules. Moreover, it created rules for only 23 classes, while the ensemble method learned rules for 234 class. Hence, there were 7-8 rules per class in classifiers created by the ensemble method, while there were 785 rules per class in the classifier of Lægreid et al.

Thus, it is possible that better results could have been obtained with the genetic reduct algorithm than with the search algorithms that were used in this study. In order to examine this hypothesis a bit further, we tried to weaken our results so that we might obtain a recall depth and a precision depth that were similar to that of Lægreid et al. This was done by changing θ for ensemble method and the pruning accuracy δ for the bottom-up method. Some of the results that were achieved are reported in Table 10.4. However, we were unable to obtain a similar detail level as these results indicate.

This suggests that more details could, perhaps, have been retained if the genetic reduct algorithm (or another less noise sensitive algorithm) had been used in the bottom-up and ensemble approaches. However, it is not certain that we would have obtained a *large* increase in the detail level if this algorithm had been used. The classifier of Lægreid et al. had a low precision, and 17 of the 23 classes in their study had a depth of 2 or 3. So, if we wanted to maintain a high precision, we would have had to give up some of the details that were obtained in their study. Thus, we might still have ended up with a classifier that predicted mostly classes at the top level (i.e., at depth 1). So, even though it is possible that some more detail could have been obtained with another search algorithm, we believe that our results are reasonable given the available data. A much larger data set comprising many more measurements (created under different experimental conditions) would have been required in order to obtain more detailed predictions.

10.3 The artificial data

The fibroblast data set was difficult for prediction of gene function. Therefore, we performed several controlled experiments in order to get a better understanding of the performance of the algorithms. These experiments are reported in this section. In particular, we demonstrate the importance of avoiding discrimination between related classes, and trading off details for precision. Moreover, we examine whether the algorithms predict too general classes such that details are lost needlessly.

10.3.1 Data material

Several different artificial data sets were created for these experiments. Two different DAGs – one small and one quite large – were constructed initially. The small DAG contained 13 classes where 7 classes were leaf classes. Five of these leaf classes had a depth of 3, and two had a depth of 2. The large DAG consisted of 52 classes with 30 leaf classes. Twenty of these leaf classes had a depth of 4. The rest had a depth of 3.

Objects were constructed in two steps. Model objects were first created and assigned to one or more leaf classes. Each model object had an information vector that consisted of 11 attributes where each attribute could be assigned one of the following values: *up*, *down*, and *const*. Objects were then created from the model objects. An object was produced as follows: The attribute-value pairs in the information vector of the model object were first copied to the information vector of the object. The object was then annotated by randomly generalizing the leaf class label(s) of the model object. For each leaf class label, a class was selected at random from a pool that consisted of the leaf class and its superclasses. The object was then labeled with the selected class.

Several objects were created for each model object. Fifteen instances of each model objects were created for the small DAG, 25 instances were created for the large DAG. These numbers may seem large, however, only a fraction of these objects were assigned to the leaf classes. The most detailed leaf classes in the small DAG had on average 5 instances of same model object. The deepest classes in the large DAG had 6.25 instances on average.

10.3.2 Results

Without errors

The bottom-up and ensemble methods were applied to both data sets. The performance measures were estimated with 10-fold cross-validation. The results are shown in Table 10.5 and Table 10.6.

The bottom-up method worked almost perfectly. All predictions were correct and all annotations were matched. However, a very small loss of details was observed. This was expected as the bottom-up method requires a minimum of objects in order to accept a rule.

Instances of the same model object had to occur in both the training set and the validation set in order for the bottom-up method to learn a rule. Objects were needed in the training set such that a rule was learned, and objects were required in the validation set such that a rule was not pruned. Hence, a leaf class needed at least 3 objects (when $\rho = 0.66$) such that 2 objects were placed in the training set and 1 object was put in the validation set. However, a higher number of objects could be required if several model objects were annotated to the same class since the random splitting, which created the training and the validation sets, could place all instances of a model object in only one of these two sets. The likelihood for this to happen depended on the number of instances that were annotated to each class. However, the situation was not unlikely when the number of instances of each model object was very low as it was in the leaf classes of our data sets.

The results seem very good given this requirement for objects since there were only 5 or 6.25 objects on average in the data set, and this number was reduced by approximately 10% in the training sets created by the 10-fold cross-validation procedure. Moreover, by setting $\rho = 0.55$, we sometimes achieved perfect results (these are not shown). However, this depended on the seed given to the random number generator, which was used when the data were split for cross-validation. Increasing the number of instances per model object to 20 for the small DAG gave perfect results as well. Thus the method did not appear to have a tendency to predict too general classes, besides the small loss that was caused by the need to fill both the training and the validation set with objects.

The performance of the ensemble method was perfect when θ was 1. However, the level of detail was reduced when θ was decreased. The loss was marginal at $\theta = 0.7$, but was clearly a concern when $\theta = 0$. At this level, the recall depth was only 0.856.

As explained in Section 8.3.2, the loss was caused by overestimation of votes in some of the rules. This could occur if two objects that were labeled with different classes had almost similar information vectors (but with some different attribute-value pairs). A rule created for a common superclass of these classes could cover both objects in this case and would have twice as many votes as the rules created for each of the two classes. So, when a new object, which was similar to one of the objects, was classified, the rule in the superclass would have a higher vote. The superclass would therefore be selected, and this resulted in a loss of details.

The flat methods were also tested on the data sets. The first of these did not predict

Method	***Search***	θ	***Match (Global)***		***Depth***	
			Recall (RA)	***Precision (RP)***	***Recall (DA)***	***Precision (DP)***
Bottom-up						
	Top-down		**1.000**±0.000	**1.000**±0.000	**0.989**±0.005	**0.989**±0.005
	Bottom-up		**1.000**±0.000	**1.000**±0.000	**0.989**±0.005	**0.989**±0.005
Ensemble						
(Normal)	***Top-down***	*1*	**1.000**±0.000	**1.000**±0.000	**1.000**±0.000	**1.000**±0.000
	Bottom-up	*1*	**1.000**±0.000	**1.000**±0.000	**1.000**±0.000	**1.000**±0.000
	Top-down	*0.7*	**1.000**±0.000	**1.000**±0.000	**0.990**±0.006	**0.990**±0.006
	Bottom-up	*0.7*	**1.000**±0.000	**1.000**±0.000	**0.990**±0.006	**0.990**±0.006
	Top-down	*0.5*	**1.000**±0.000	**1.000**±0.000	**0.988**±0.006	**0.977**±0.006
	Bottom-up	*0.5*	**1.000**±0.000	**1.000**±0.000	**0.988**±0.006	**0.977**±0.006
	Top-down	*0*	**1.000**±0.000	**1.000**±0.000	**0.958**±0.014	**0.949**±0.015
	Bottom-up	*0*	**1.000**±0.000	**1.000**±0.000	**0.958**±0.014	**0.949**±0.015
Ensemble						
(Approx)	***Top-down***	*1*	**1.000**±0.000	**1.000**±0.000	**1.000**±0.000	**1.000**±0.000
	Bottom-up	*1*	**1.000**±0.000	**1.000**±0.000	**1.000**±0.000	**1.000**±0.000
	Top-down	*0.7*	**1.000**±0.000	**1.000**±0.000	**0.990**±0.006	**0.990**±0.006
	Bottom-up	*0.7*	**1.000**±0.000	**1.000**±0.000	**0.990**±0.006	**0.990**±0.006
	Top-down	*0.5*	**1.000**±0.000	**1.000**±0.000	**0.988**±0.006	**0.977**±0.006
	Bottom-up	*0.5*	**1.000**±0.000	**1.000**±0.000	**0.988**±0.006	**0.977**±0.006
	Top-down	*0*	**1.000**±0.000	**1.000**±0.000	**0.958**±0.014	**0.949**±0.015
	Bottom-up	*0*	**1.000**±0.000	**1.000**±0.000	**0.958**±0.014	**0.949**±0.015
Flat						
(Voting)	***Top-down***		**0.000**±0.000	**0.000**±0.000	**0.000**±0.000	**0.000**±0.000
	Bottom-up		**0.000**±0.000	**0.000**±0.000	**0.000**±0.000	**0.000**±0.000
Flat						
(No voting)	***Top-down***		**1.000**±0.000	**1.000**±0.000	**1.000**±0.000	**0.847**±0.010
	Bottom-up		**1.000**±0.000	**1.000**±0.000	**1.000**±0.000	**0.847**±0.010

Table 10.5: *Results for the small DAG without errors.* The training accuracy γ was set to 1 for all algorithms. The bottom-up algorithm was executed with top level pruning and $\rho = 0.66$, $\delta = 1$, and $\sigma = 1$ (similar results were also obtained without top level pruning).

Method	***Search***	*θ*	***Match (Global)*** ***Recall (RA)***	***Precision (RP)***	***Depth*** ***Recall (DA)***	***Precision (DP)***
Bottom-up						
	Top-down		**1.000**±0.000	**1.000**±0.000	**0.996**±0.001	**0.994**±0.002
	Bottom-up		**1.000**±0.000	**1.000**±0.000	**0.996**±0.001	**0.996**±0.001
Ensemble						
(Normal)	***Top-down***	*1*	**1.000**±0.000	**1.000**±0.000	**1.000**±0.000	**1.000**±0.000
	Bottom-up	*1*	**1.000**±0.000	**1.000**±0.000	**1.000**±0.000	**1.000**±0.000
	Top-down	*0.7*	**1.000**±0.000	**1.000**±0.000	**0.964**±0.004	**0.957**±0.005
	Bottom-up	*0.7*	**1.000**±0.000	**1.000**±0.000	**0.962**±0.005	**0.954**±0.006
	Top-down	*0.5*	**1.000**±0.000	**1.000**±0.000	**0.922**±0.008	**0.915**±0.009
	Bottom-up	*0.5*	**1.000**±0.000	**1.000**±0.000	**0.923**±0.008	**0.915**±0.009
	Top-down	*0*	**1.000**±0.000	**1.000**±0.000	**0.856**±0.013	**0.843**±0.013
	Bottom-up	*0*	**1.000**±0.000	**1.000**±0.000	**0.860**±0.012	**0.853**±0.011
Ensemble						
(Approx)	***Top-down***	*1*	**1.000**±0.000	**1.000**±0.000	**1.000**±0.000	**1.000**±0.000
	Bottom-up	*1*	**1.000**±0.000	**1.000**±0.000	**1.000**±0.000	**1.000**±0.000
	Top-down	*0.7*	**1.000**±0.000	**1.000**±0.000	**0.964**±0.004	**0.963**±0.004
	Bottom-up	*0.7*	**1.000**±0.000	**1.000**±0.000	**0.956**±0.006	**0.954**±0.007
	Top-down	*0.5*	**1.000**±0.000	**1.000**±0.000	**0.922**±0.008	**0.916**±0.009
	Bottom-up	*0.5*	**1.000**±0.000	**1.000**±0.000	**0.929**±0.008	**0.923**±0.009
	Top-down	*0*	**1.000**±0.000	**1.000**±0.000	**0.857**±0.013	**0.850**±0.013
	Bottom-up	*0*	**1.000**±0.000	**1.000**±0.000	**0.876**±0.012	**0.869**±0.011
Flat						
(Voting)	***Top-down***		**0.000**±0.000	**0.000**±0.000	**0.000**±0.000	**0.000**±0.000
	Bottom-up		**0.000**±0.000	**0.000**±0.000	**0.000**±0.000	**0.000**±0.000
Flat						
(No voting)	***Top-down***		**1.000**±0.000	**1.000**±0.000	**1.000**±0.000	**0.819**±0.007
	Bottom-up		**1.000**±0.000	**1.000**±0.000	**1.000**±0.000	**0.819**±0.007

Table 10.6: *Results for the large DAG without errors.* The training accuracy γ was set to 1 for all algorithms. The bottom-up algorithm was executed with top level pruning and $\rho = 0.66$, $\delta = 1$, and $\sigma = 1$ (similar results were also obtained without top level pruning).

Method	*Search*	*Match (Global)*		*Depth*	
		Recall (RA)	*Precision (RP)*	*Recall (DA)*	*Precision (DP)*
Bottom-up					
	Top-down	**1.000**±0.000	**0.993**±0.007	**0.989**±0.006	**0.989**±0.006
	Bottom-up	**1.000**±0.000	**0.995**±0.005	**0.943**±0.012	**0.921**±0.019
Ensemble					
(Normal)	*Top-down*	**1.000**±0.000	**1.000**±0.000	**0.959**±0.013	**0.959**±0.013
	Bottom-up	**1.000**±0.000	**1.000**±0.000	**0.997**±0.003	**0.998**±0.003
Ensemble					
(Approx)	*Top-down*	**1.000**±0.000	**1.000**±0.000	**1.000**±0.000	**1.000**±0.000
	Bottom-up	**1.000**±0.000	**1.000**±0.000	**0.997**±0.003	**0.998**±0.003
Flat					
(Voting)	*Top-down*	**0.446**±0.042	**0.949**±0.026	**0.863**±0.047	**0.854**±0.045
	Bottom-up	**0.373**±0.054	**1.000**±0.000	**0.862**±0.036	**0.854**±0.035
Flat					
(No voting)	*Top-down*	**0.684**±0.021	**0.785**±0.043	**0.877**±0.026	**0.851**±0.020
	Bottom-up	**0.462**±0.052	**0.664**±0.055	**0.867**±0.033	**0.838**±0.028

Table 10.7: *Results for the small DAG with irrelevant attributes.* The training γ was set to 1 for all algorithms. The bottom-up algorithm was executed with top level pruning and $\rho = 0.66$, $\delta = 1$, and $\sigma = 1$ (similar results were also obtained without top level pruning). θ was set to 1 for the ensemble method.

any classes just as it did on the fibroblast data. The other flat method worked quite well – its performance was almost as good as those of the bottom-up and ensemble methods. The reason for this was that there were no irrelevant attributes in the data set. The search algorithms had no opportunity to make mistakes. So even though it tried to discriminate between similar classes, it found rules that covered the model vectors. Its performance was therefore quite good.

With irrelevant attributes

The data sets without errors correspond to an ideal situation where each attribute contributes to the discernibility of the classes. Such a situation will rarely occur in practice. A more realistic data set will contain noise. In particular, there will be attributes that do not contribute to the discernibility of the classes. We called such attributes irrelevant attributes.

In order to make the data sets more realistic, we added 11 irrelevant attributes to each information vector such that each vector consisted of a total 22 attributes. The values of the irrelevant attributes were generated at random when an object was created. The same three values that were used for the relevant attributes were also assigned to these attributes.

Method	***Search***	***Match (Global)***		***Depth***	
		Recall (RA)	***Precision (RP)***	***Recall (DA)***	***Precision (DP)***
Bottom-up					
	Top-down	**1.000**±0.000	**0.999**±0.001	**0.994**±0.002	**0.992**±0.003
	Bottom-up	**1.000**±0.000	**0.999**±0.001	**0.990**±0.002	**0.986**±0.002
Ensemble					
(Normal)	***Top-down***	**1.000**±0.000	**1.000**±0.000	**0.999**±0.001	**0.999**±0.001
	Bottom-up	**0.999**±0.001	**0.999**±0.001	**0.996**±0.001	**0.996**±0.001
Ensemble					
(Approx)	***Top-down***	**1.000**±0.000	**1.000**±0.000	**1.000**±0.000	**1.000**±0.000
	Bottom-up	**0.999**±0.001	**0.999**±0.001	**0.996**±0.001	**0.996**±0.001
Flat					
(Voting)	***Top-down***	**0.477**±0.018	**1.000**±0.000	**0.850**±0.020	**0.844**±0.021
	Bottom-up	**0.258**±0.012	**1.000**±0.000	**0.797**±0.023	**0.798**±0.023
Flat					
(No voting)	***Top-down***	**0.657**±0.015	**0.777**±0.018	**0.862**±0.015	**0.836**±0.019
	Bottom-up	**0.444**±0.011	**0.532**±0.015	**0.815**±0.022	**0.809**±0.018

Table 10.8: *Results for the large DAG with irrelevant attributes.* The training accuracy γ was set to 1 for all algorithms. The bottom-up algorithm was executed with top level pruning and $\rho = 0.66$, $\delta = 1$, and $\sigma = 1$ (similar results were also obtained without top level pruning). θ was set to 1 for the ensemble method.

The results from these data sets are given in Table 10.7 and Table 10.8 (computed as before with 10-fold cross-validation). The results for the bottom-up and ensemble methods were very good. There were some very small errors, e.g., the bottom-up method had an *RP* of 0.993 and 0.995. These errors were not significant as reflected by the associated standard errors. They were estimation errors that were introduced by unfortunate splits made by the cross-validation procedure. This was confirmed in several cases by changing the seed of the random number generator that was used by the cross-validation procedure. In all of these cases, we were able to obtain perfect results[7].

Most of the results were similar to those obtained in the last section. However, the bottom-up method had a lower recall and precision depth when bottom-up search was used on the small DAG. This loss occurred since the search algorithm would create a rule with an irrelevant attribute at some occasions and such rules would be pruned since they did not a have a high accuracy.

A similar loss in detail level was observed for the ensemble method with top-down search. In this case, it seemed that the top-down search algorithm could be confused by random properties in the data when it learned rules for the complement.

[7]These results are not shown as we have chosen to create all of the results in this chapter with the same seed such that the results are more comparable.

The positive set for complement contained many objects with different values for the relevant attribute. None of these values were very frequent. A value of an irrelevant attribute could actually be slightly more frequent in some cases due to the random properties of the data. It would thus have a higher score and be selected by the search algorithm. The resulting rule would not make accurate predictions and could cause the votes against a class to be overestimated. A (leaf) class could therefore be rejected even though it should not, and this resulted in some loss of details.

The results obtained with the two flat methods were clearly much worse than those of the bottom-up and ensemble methods. The search algorithms tried to separate related classes. This resulted in rules that were based on the irrelevant attributes since none of the relevant attributes could separate objects that were created from the same model object and annotated to related (but different) classes. The predictions made by these methods were therefore very poor. This clearly demonstrates that a DAG learning algorithm should not attempt to discern between related classes.

With inconsistencies

The ability of the algorithms to trade off details for precision was also examined. This was done by adding inconsistencies to the original data sets with no errors. An inconsistency was created by assigning model objects with identical information vectors to different leaf classes. Each model object was assigned to one leaf class, and the leaf classes were selected such that they had at least one superclass in common besides the root. Objects were then created from these model objects as before, except that the class labels of these objects were not generalized. All of these objects were thus assigned to leaf classes.

In biological terms, one could think of an inconsistency as a group of genes that were similarly expressed, but were labeled to different (leaf) classes. A classifier that tried to predict one or more of the leaf class(es) for the inconsistent objects (i.e, the similar expressed genes) would obviously make errors. The objects could not be discerned, and each object was assigned to only one class. Hence, a prediction to one leaf class would not be correct for all of the inconsistent objects. However, if the classifier was able to trade off details for precision such that it predicted the common superclass instead, no errors would occur.

Several inconsistencies were created for both DAGs. These were made with different information vectors. The classes, which were assigned to the model objects, were selected such that the depth of the common superclass varied. Given the properties of the data sets, we could compute the optimal recall/precision depth if the learning algorithm identified the common superclasses correctly. This was 0.846 for the small DAG and 0.901 for the large DAG. Moreover, if an algorithm made no redundant predictions (i.e., made no predictions that were related) and predicted a leaf class for the inconsistent objects, it would, in the worst case, have a precision of 0.765 for the small DAG and a precision of 0.824 for the large DAG.

The algorithms were tested on the data sets with 10-fold cross-validation. These results are given in Table 10.9 and Table 10.10. All of our methods worked more or

Method	*Search*	*Match (Global)*		*Depth*	
		Recall (RA)	*Precision (RP)*	*Recall (DA)*	*Precision (DP)*
Bottom-up					
	Top-down	**1.000**±0.000	**1.000**±0.000	**0.823**±0.018	**0.795**±0.012
	Bottom-up	**1.000**±0.000	**1.000**±0.000	**0.828**±0.019	**0.801**±0.018
Ensemble					
(Normal)	*Top-down*	**1.000**±0.000	**1.000**±0.000	**0.845**±0.022	**0.845**±0.022
	Bottom-up	**1.000**±0.000	**1.000**±0.000	**0.845**±0.022	**0.845**±0.022
Ensemble					
(Approx)	*Top-down*	**1.000**±0.000	**1.000**±0.000	**0.845**±0.022	**0.845**±0.022
	Bottom-up	**1.000**±0.000	**1.000**±0.000	**0.845**±0.022	**0.845**±0.022
Flat					
(Voting)	*Top-down*	**0.000**±0.000	**0.000**±0.000	**0.000**±0.000	**0.000**±0.000
	Bottom-up	**0.000**±0.000	**0.000**±0.000	**0.000**±0.000	**0.000**±0.000
Flat					
(No voting)	*Top-down*	**1.000**±0.000	**0.871**±0.019	**1.000**±0.000	**0.861**±0.010
	Bottom-up	**1.000**±0.000	**0.871**±0.019	**1.000**±0.000	**0.861**±0.010

Table 10.9: *Results for the small DAG with inconsistencies.* The training accuracy γ was set to 1 for all algorithms. The bottom-up algorithm was executed with top level pruning and $\rho = 0.66$, $\delta = 1$, and $\sigma = 1$ (similar results were also obtained without top level pruning). θ was set to 1 for the ensemble method. The optimal recall/precision depth was 0.846 for this data set.

less perfectly. A small loss of details was visible for the bottom-up. However, this was of the same magnitude as observed previously and occurred most likely because this method needed a few more objects in order to learn and accept a rule.

The first flat method did not predict any classes at all. The performance of the second flat method was similar to its previous performance when the data sets did not contain any errors. However, the precision was lower because the algorithm predicted the leaf classes for the inconsistent objects. Still, it was higher than expected. This happened because the algorithm made a lot of redundant predictions. Basically, a leaf class and all of its superclasses were predicted for the same annotation. The number of correct predictions was therefore quite high compared to the number of incorrect predictions, and this resulted in an inflated precision. However, inspection of the prediction revealed that it made all the incorrect predictions that were possible.

With irrelevant attributes and inconsistencies

Finally, the two kinds of errors were combined in order to examine how the algorithms handled their combined effect. 11 irrelevant attributes were added to the data sets with the inconsistencies. These attributes were assigned to the information vectors

Method	*Search*	*Match (Global)*		*Depth*	
		Recall (RA)	*Precision (RP)*	*Recall (DA)*	*Precision (DP)*
Bottom-up					
	Top-down	**1.000**±0.000	**1.000**±0.000	**0.890**±0.005	**0.844**±0.007
	Bottom-up	**1.000**±0.000	**1.000**±0.000	**0.899**±0.005	**0.869**±0.007
Ensemble (Normal)	*Top-down*	**1.000**±0.000	**1.000**±0.000	**0.892**±0.007	**0.890**±0.007
	Bottom-up	**1.000**±0.000	**1.000**±0.000	**0.892**±0.007	**0.890**±0.007
Ensemble (Approx)	*Top-down*	**0.996**±0.003	**0.996**±0.003	**0.895**±0.006	**0.893**±0.006
	Bottom-up	**1.000**±0.000	**1.000**±0.000	**0.892**±0.007	**0.890**±0.007
Flat (Voting)	*Top-down*	**0.000**±0.000	**0.000**±0.000	**0.000**±0.000	**0.000**±0.000
	Bottom-up	**0.000**±0.000	**0.000**±0.000	**0.000**±0.000	**0.000**±0.000
Flat (No voting)	*Top-down*	**1.000**±0.000	**0.937**±0.004	**1.000**±0.000	**0.832**±0.003
	Bottom-up	**1.000**±0.000	**0.937**±0.004	**1.000**±0.000	**0.832**±0.003

Table 10.10: *Results for the large DAG with inconsistencies.* The training accuracy γ was set to 1. The bottom-up algorithm was run with top level pruning and $\rho = 0.66$, $\delta = 1$, and $\sigma = 1$ (similar results were obtained without top level pruning). θ was set to 1 for the ensemble method. The optimal DA and DP was 0.901 for this data set.

of both consistent and inconsistent objects. The inconsistent objects were therefore not inconsistent in the rough set sense of the word as they did not share the same elementary set. However, they did share the same values for the relevant attributes. These objects could therefore not be separated by any rule that was based only on the relevant attributes. A rule based on the irrelevant attributes could possibly separate them. However, it would not make any accurate predictions since the values of these attributes were generated at random. Obviously, this made it much more difficult for an algorithm to recognize that these objects should be predicted to their common superclass and not to the leaf classes.

Note that this kind of inconsistency could occur in a real life data set if some genes were really similarly expressed (and differently labeled), but their expression profiles (i.e., information vector) were not precisely identical due to some noisy attribute values. The noisy attribute values would not be sufficient to predict the objects to the detailed leaf classes. Thus, a learning algorithm would have to predict the common superclass in this case.

The results are given in Table 10.11 and Table 10.12. The bottom-up and ensemble methods performed clearly much better than the flat methods. However, their performance was not perfect in these data sets. The ensemble method made several incorrect predictions and was unable to predict all annotations when θ was 1. This

Method	*Search*	θ	σ	*Top lev. pru.*	*Match (Global)* *Recall (RA)*	*Precision (RP)*	*Depth* *Recall (DA)*	*Precision (DP)*
Bottom-up								
	Top-down		*1*	***Yes***	**1.00**±0.00	**0.98**±0.01	**0.84**±0.02	**0.79**±0.01
	Bottom-up		*1*	***Yes***	**1.00**±0.00	**0.97**±0.01	**0.82**±0.02	**0.80**±0.02
	Top-down		*3*	***Yes***	**0.97**±0.02	**1.00**±0.00	**0.68**±0.02	**0.66**±0.02
	Bottom-up		*3*	***Yes***	**0.97**±0.02	**0.99**±0.01	**0.70**±0.02	**0.67**±0.02
	Top-down		*3*	***No***	**1.00**±0.00	**1.00**±0.00	**0.69**±0.02	**0.67**±0.02
	Bottom-up		*3*	***No***	**1.00**±0.00	**0.99**±0.01	**0.69**±0.02	**0.67**±0.02
Ensemble								
(Normal)	***Top-down***	*1*			**0.89**±0.03	**0.87**±0.03	**0.93**±0.01	**0.93**±0.01
	Bottom-up	*1*			**0.96**±0.02	**0.95**±0.02	**0.90**±0.01	**0.90**±0.01
	Top-down	*0.7*			**1.00**±0.00	**0.99**±0.01	**0.81**±0.02	**0.81**±0.02
	Bottom-up	*0.7*			**1.00**±0.00	**1.00**±0.00	**0.81**±0.02	**0.81**±0.02
Ensemble								
(Approx)	***Top-down***	*1*			**0.89**±0.02	**0.88**±0.02	**0.92**±0.01	**0.92**±0.01
	Bottom-up	*1*			**0.96**±0.02	**0.96**±0.02	**0.90**±0.02	**0.90**±0.02
	Top-down	*0.7*			**1.00**±0.00	**0.99**±0.01	**0.81**±0.02	**0.81**±0.02
	Bottom-up	*0.7*			**1.00**±0.00	**1.00**±0.00	**0.82**±0.02	**0.82**±0.02
Flat								
(Voting)	***Top-down***				**0.36**±0.04	**0.79**±0.07	**0.86**±0.03	**0.85**±0.02
	Bottom-up				**0.29**±0.04	**0.88**±0.04	**0.92**±0.02	**0.92**±0.02
Flat								
(No voting)	***Top-down***				**0.54**±0.04	**0.69**±0.05	**0.87**±0.02	**0.85**±0.02
	Bottom-up				**0.40**±0.03	**0.53**±0.03	**0.88**±0.03	**0.87**±0.03

Table 10.11: *Results for the small DAG with irrelevant attributes and inconsistencies.* The training accuracy γ was set to 1 for all algorithms. The bottom-up algorithm was executed with $\rho = 0.66$ and $\delta = 1$. The optimal recall/precision depth was 0.846 for this data set.

was quite evident when top-down search was used.

These errors occurred since the inconsistent objects did not belong to the same elementary set and could not be identified by the rough set operators. This meant that both the positive and the negative set could contain objects with the same values for the relevant attributes when rules were learned for a class (or for the complement of a class). The search algorithms would therefore try to separate these objects and create rules with some irrelevant attributes since they could not be discerned by the relevant attributes. These rules did not make accurate predictions and had a low support. The votes made for and against a leaf class would therefore be incorrect. Typically, both the votes for and against a class would be underestimated. A random prediction made by one of these rules, could therefore fool the prediction algorithm into accepting a leaf class – even though it should not be accepted. Moreover, the votes for the superclasses of this leaf class would also be underestimated since the

Method	*Search*	θ	σ	*Match (Global)*		*Depth*	
				Recall (RA)	*Precision (RP)*	*Recall (DA)*	*Precision (DP)*
Bottom-up							
	Top-down		*1*	**1.00**±0.00	**0.96**±0.00	**0.90**±0.01	**0.85**±0.01
	Bottom-up		*1*	**1.00**±0.00	**0.98**±0.00	**0.89**±0.01	**0.86**±0.01
	Top-down		*3*	**1.00**±0.00	**1.00**±0.00	**0.85**±0.00	**0.81**±0.01
	Bottom-up		*3*	**1.00**±0.00	**1.00**±0.00	**0.85**±0.00	**0.83**±0.01
Ensemble							
(Normal)	***Top-down***	*1*		**0.93**±0.01	**0.92**±0.01	**0.95**±0.00	**0.95**±0.00
	Bottom-up	*1*		**0.95**±0.01	**0.95**±0.01	**0.94**±0.00	**0.94**±0.00
	Top-down	*0.7*		**1.00**±0.00	**0.99**±0.00	**0.86**±0.00	**0.85**±0.01
	Bottom-up	*0.7*		**1.00**±0.00	**0.99**±0.00	**0.86**±0.01	**0.85**±0.01
Ensemble							
(Approx)	***Top-down***	*1*		**0.93**±0.01	**0.90**±0.01	**0.96**±0.00	**0.96**±0.00
	Bottom-up	*1*		**0.95**±0.01	**0.93**±0.01	**0.95**±0.00	**0.94**±0.00
	Top-down	*0.7*		**1.00**±0.00	**0.98**±0.00	**0.86**±0.00	**0.85**±0.01
	Bottom-up	*0.7*		**1.00**±0.00	**0.99**±0.00	**0.87**±0.01	**0.86**±0.01
Flat							
(Voting)	***Top-down***			**0.45**±0.02	**0.86**±0.02	**0.87**±0.01	**0.86**±0.01
	Bottom-up			**0.25**±0.01	**0.85**±0.02	**0.85**±0.01	**0.84**±0.01
Flat							
(No voting)	***Top-down***			**0.62**±0.02	**0.70**±0.03	**0.89**±0.01	**0.86**±0.00
	Bottom-up			**0.42**±0.02	**0.50**±0.02	**0.84**±0.01	**0.83**±0.01

Table 10.12: *Results for the large DAG with irrelevant attributes and inconsistencies.* The training accuracy γ was set to 1 for all algorithms. The bottom-up algorithm was executed with top level pruning and $\rho = 0.66$ and $\delta = 1$ (similar results were also obtained without top level pruning). The optimal recall/precision depth was 0.901 for this data set.

votes for a superclass were based entirely on the votes from its subclasses when $\theta = 1$. The prediction algorithm would therefore select the leaf class over the superclass, and this resulted in an incorrect prediction.

However, the performance was almost perfect when θ was set 0.7. The precision was very close to perfect, and the loss of details was very small. In this case, the votes for a superclass were not only based on the votes cast for it subclasses, but also on the votes cast by the rule(s) that predicted the superclass. The votes for a superclass would therefore be estimated more correctly so that the prediction algorithm would select the superclass over a leaf class, which had been incorrectly accepted.

The bottom-up method was more robust than the ensemble method. It maintained the detail level quite well when the pruning support σ was set to 1. All annotations were predicted and only a very few predictions were incorrect in this case.

These errors occurred because a few poor rules slipped through the pruning. The search algorithms also tried in this case to separate between inconsistent objects when rules were learned for a leaf class with such objects. The rules created for these objects therefore had descriptors with irrelevant attributes. These rules did not cover many objects in the validation set so that their accuracy and support was low. Hence, they were usually removed by the pruning algorithm. Nevertheless, on some rare occasions, a rule had a high accuracy and was accepted, and these rules created the errors.

We were able to remove these errors in most cases by raising support σ to 3. However, this meant that the algorithms now needed more objects in order to learn and accept a rule. Some loss of details was therefore observed – especially for the small DAG where objects were less abundant. This was higher than the loss of precision that was observed when $\sigma = 1$. So, the results for $\sigma = 1$ seemed better. The precision was quite close to the optimal value. σ should thus be kept at 1 – at least when the number of objects is sparse.

10.4 Conclusion

The experiments with the artificial data sets demonstrate that the bottom-up and ensemble methods deal appropriately with the issues introduced by the DAG. The methods do not discern between related classes. The rules created by them have a much higher quality than the rules of corresponding flat methods. Thus, the predictions made by these methods are much more accurate. The bottom-up and ensemble methods are able to learn good classifiers when the annotations have a varying detail level and the data associated with each class is relatively sparse. The classes of the model objects in these experiments were randomly generalized so that the annotations had a different detail level, and only a few (5-6) objects were assigned to each class. The algorithms still managed to produce high quality classifiers. Both methods seem to handle the trade-off between detail level and precision effectively. When objects are labeled with classes that cannot be predicted accurately, they identify the most specific classes that may be predicted.

It is possible that a DAG learning method may lose details that could have been retained. However, this does not appear to be a problem for the bottom-up method. It retained the detail level quite well in all of the artificial experiments, and it did not show a tendency to predict too general classes. Only a very small loss of details was witnessed in our results. This was expected as the method requires a minimum of objects in order to split the data into a training set for learning and a validation set for pruning. So if the examples are very sparse, a small loss may be observed. However, we believe that this should not be any problem in a real life situation.

The ensemble method does not need to split the data. Its need for objects is thus slightly lower than the bottom-up method. This was witnessed in the experiments where this method was often able to obtain a perfect score. It therefore has a small advantage over the bottom-up method when the data are very sparse. However, this advantage may not be important in a practical situation.

This algorithm also has a disadvantage. The rules in the superclasses may overestimate the votes for a class such that too general classes are predicted. This is not a problem when $\theta = 1$ since the votes are estimated precisely in this case, but a significant loss may occur when $\theta = 0$. Unfortunately, the algorithm is sensitive to noise when $\theta = 1$. This was demonstrated in the last experiment. Adjusting θ to a position close to 1 seems to correct this situation. However, the method appears less robust than the bottom-up method with regard to noise.

The results for both versions of the ensemble method were quite similar. However, the approximate version needs fewer rules and is much faster since it does not learn dissenting rules. Hence, this version may be quite useful even though its estimate of the negative support is not accurate.

The results on the fibroblast data also show that the bottom-up method and the ensemble method work much better than a corresponding flat method. The predictions obtained on this data set are accurate. However, they are very general such that they do not contribute much biological knowledge. Still, it seems that this is the best that can be achieved with the available data set when the top-down or the bottom-up search algorithm is used. It is possible that another search algorithm may produce more detailed predictions. However, the main problem seems to be that the fibroblast data set is insufficient for making both detailed and accurate predictions. Our methods should therefore be applied on a much larger data set.

Conclusions and Future Research

11

11.1 Main results and contributions

This thesis lies in borderland between several different disciplines. On the one hand, we have molecular biology and microarray technology that raise the problems. On the other, we have computer science and knowledge discovery/data mining that provide the solutions together with statistics and mathematics. These fields represent different kinds of knowledge that must be unified in order to provide solutions for microarray analysis. Such unification is obviously a complex task. The thesis addresses this challenge by focusing on several typical microarray studies and presenting solutions where relevant contributions from these disciplines are brought together and novel methods are developed.

In particular, the thesis considers classification problems for two kinds of microarray studies. These may be described according to the objects that are classified. In the first kind of study, tumor samples are classified, and in the second kind of study, genes are classified.

We have explained the main issues involved in these analyses and developed general methods that solve them. These methods comprise not only learning, but also preprocessing and evaluation. The preprocessing methods are perhaps not original in the sense of particular solutions, but are novel in their composition with the learning methods. The methods have been tested on experimental and artificial data. In this process, several software tools have been implemented. These are currently prototypes, but can easily be developed into more user friendly tools.

There are many issues that remain before we have fully automated systems where a biologist can push a button and receive a final analysis. Developing such tools is a long and complex process, and we have probably only scratched the surface in some cases. Hence, there are a lot of opportunities for further research. This will be discussed in detail in Section 11.3. The next sections summarize the results that have been obtained.

Classification of samples

This kind of analysis is important in cancer studies where supervised learning may be applied to determine the tumor subgroup and other clinical parameters. Prediction of such parameters from microarrays may allow earlier detection of cancer and more effective treatment.

The main computational issue in this analysis is the imbalance between the number of genes and the number of samples. The number of genes is typically much larger than the number of samples. A supervised learning system, on the other hand, expects that the number of samples is larger than the number of genes.

Therefore, we have introduced a general methodology based on rough sets and bootstrapping where a set of genes is selected by a bootstrapping algorithm before the data are given to a rough set learning algorithm. This approach has been tested on a data set of gastric tumors, and its ability to develop high quality classifiers has been demonstrated. In particular, we have shown that the feature selection step is required in order to obtain good classifiers. Our results also indicate that rough set learning algorithms may work better than discriminant analysis on this kind of data. Moreover, it has been shown that high quality classifiers for several different parameters can be developed from the same data set.

Classifiers have been built for a total of 6 different clinical parameters. The connections between the parameters and the genes used in these classifiers have been compared to the establish knowledge in the biomedical literature. Some of these connections could be confirmed. However, many of these connections have not yet been described in the literature and may provide valuable hypotheses for further biological research.

The gastric cancer data set is rather small. Several of the clinical parameters have a very skew class distribution where only 3 or 4 objects belong to the minority class. With that few objects, it is possible that good classifiers may be obtained just by chance. Hence, the genes used by these classifiers may not be biologically connected to the clinical parameters. The results for the parameters with skew class distribution must thus be interpreted with caution.

Classification of genes in an ontology

This kind of analysis is performed in order to determine the function of genes and is based on the assumption that co-expressed genes share the same function. A major concern in this analysis is the class descriptions, which are associated with the genes. These descriptions are typically organized in an ontology. An ontology introduces several problems that are not considered by an ordinary learning algorithm:

- The ontology defines relationships between the classes.
- The relationships imply that the classes have different levels of detail, but a gene may be annotated to any class. Thus, the detail level of the annotations may vary.

- The data associated with each class may be very sparse.
- It may be impossible to obtain both detailed and accurate predictions. An algorithm may have to trade off details for precision.

Moreover, a gene may be annotated to more than one class. A learning algorithm for this task must therefore be able to predict several different classes for each object.

Several tools have been presented to cope with these problems. A general rough set framework for a DAG have been introduced. Several boundary sets and approximations have been defined, and their properties have been examined. Some special cases such as trees and well-defined DAGs have also been considered.

Using this framework, two novel algorithms have been defined that address the issues introduced by the DAG.

- **The bottom-up method**: This approach also attempts to learn rules for each class and does not discern between related classes. The scarcity of the data is handled in the same way as ensemble method since the algorithm moves objects upwards. However, the trade-off is handled differently. Each rule is tested and removed if it is inaccurate. The algorithm will, in that case, move the objects, which are covered by the rule, to a more general class. It will then try to cover these objects again when it learns rules for this more general class. Hence, the trade-off is determined directly by the learning algorithm in this approach.
- **The ensemble method**: This algorithm does not discern between related classes. It learns rules for each class so that an annotation in the training data will be predicted regardless of the detail level. The algorithm attempts to counteract the scarcity of the data by moving objects upwards. Both objects that are labeled to a class and objects that are labeled to the subclasses are used when it learns rules for a class. The trade-off is mainly handled through a voting procedure that attempts to minimize the loss of details and precision.

Performance measures are needed in order to determine the quality of a classifier. However, standard performance measures for supervised learning do not apply to the DAG. A gene may also have several annotations, and several predictions can be made for each gene. Performance measures such as accuracy and AUC expect that each object is labeled with only one class and that only one class is predicted for each object. Hence, new performance measures are required. We have therefore introduced several new measures. Recall and precision assess the share of annotations that are matched by some prediction and the share of predictions that match some annotation. These measures have been used for a long time in information retrieval. So, even though they were developed independently in this work, these measure are not entirely novel. However, they have not been used in context of rough sets previously and are important in this field since rough set theory allows multiple classes per object. Hence, the contribution of this work is to have introduced these measures in rough sets and in our problem domain (i.e., prediction of gene function from gene expression data). Moreover, the detail level of the annotations and predictions must also be measured.

Two novel measures that accompany recall and precision have also been defined. The recall depth and the precision depth are based on the maximal relative depth between two classes and measure how well each annotation is reproduced and how well each prediction matches the annotations. We believe that these four measures together should give a good assessment of the performance of a DAG learning algorithm.

Using these measures, we have tested the algorithms on both experimental and artificial data. The results on these data sets show that both the bottom-up and ensemble methods work much better than a corresponding flat method. The results from the artificial data demonstrate that the algorithms solve the issues that we have identified in the DAG. They do not discern between related classes such that the rules have a higher quality than the rules of a flat approach. They can handle annotations of varying detail level and classes with relatively sparse data. The bottom-up method seem slightly more robust than the ensemble method with respect to the trade-off. However, it requires a bit more data as the objects have to be split into a training sample and a validation sample. Still, both methods appear to manage the trade-off quite effectively.

There are many ontologies that describe the function of genes. This thesis has focused on the Gene Ontology. This is probably the most comprehensive ontology that is currently available for gene function. Its semantics has been discussed and some difficulties have been pointed out. The semantic rules defined by the GO Consortium specify that an instance inherits all parts of their parents. However, this inheritance rule does not appear to be implemented in the current ontology. Thus, we have not applied this rule in our experiments and have assumed that the ontology is DCA-compliant. We have, nevertheless, provided a transformation algorithm and shown that it maintains the relationships between the classes. This algorithm may be used to transform an ontology, which implements the inheritance rule, into a DCA-compliant DAG. This means that our learning methods will also be applicable if the inheritance rule should be implemented in the future. Hence, these methods are useful in either case.

Although microarray data have been the motivation for introducing these methods, they have importance far beyond microarray data. Gene function can be predicted from other kinds of data such as sequence data and Medline abstracts. Adapting our methods to such data should be fairly easy (see Section 11.3). Ontologies are also used in many other domains. One such domain is text classification where our methods should be applicable.

11.2 Related work with regard to DAG learning

The survey in Chapter 3 describes related work on classification of gene expression data. However, our work on DAG learning introduces new algorithms and methods that have importance beyond microarray data. As contributions to machine learning and rough set theory, our work should be compared to similar work in those fields.

Related work in text classification

Learning in structured classes has largely been ignored in machine learning and rough set theory. However, there is some recent work that should be mentioned. Several studies on learning of hierarchical classes for text classification have been reported. In this domain classifiers are built in order to predict categories for documents, and these categories may form a tree or a DAG.

The first study on document classification in a tree seems to have been made by Koller and Sahami [87]. They propose a top-down approach where the classification problem is decomposed into smaller subproblems. A Bayesian classifier is learned for each class in the hierarchy so that the classifier for class c predicts the immediate subclasses of c. A document is then classified by traversing the tree. The root classifier predicts first one of the immediate subclasses of the root. This subclass is visited, and one of its immediate subclasses is predicted. This traversal continues until a leaf class is reached, and this leaf class is the result of the prediction process. The approach is tightly integrated with a feature selection scheme so that different features may be associated with different levels in the tree.

This method seems appealing from a computational perspective as it divides the problem into simpler subproblems. However, it is quite different from our methods, and it does not handle problems such as the scarcity of the data or the trade-off. The objects appear to be labeled to leaf classes. Predictions are only made to leaf classes, and only one class is predicted for each document.

Moreover, their approach is based on an assumption that different features are associated with each level in the hierarchy. For example, some sibling classes may have some distinguishing properties that may be used by the classifier at the immediate superclass as well as some common properties that should be used by the classifier at the next superclass (i.e., the immediate superclass of the immediate superclass). If this assumption does not hold, such that the sibling classes have no common features, the classifier at the next superclass will have to use the same feature as the classifier at the immediate superclass. In this case, we may not expect any improvements. This is quite evident in their results. They are only able show significant improvements over a flat approach when a few features are selected for each classifier, and even in this case the improvements are quite modest. Mitchell [117] has also shown analytically that an optimal version of their approach is equivalent to a flat classifier if the same features are used at each level. Hence, the utility of their approach seems limited.

Similar approaches have been investigated by several other authors. Koller and Sahami's approach is greedy as it selects the subclass with the highest probability at non-leaf classes. An optimal version, which apparently performs a best first-search, had been given by Chakrabarti et al. [23]. In this approach several predictions can be made for each object. Dumais and Chen [41] have used a similar approach with support vector machines. There are also two different approaches that implement this idea through a neural network architecture [190, 146]. Both are inspired by a method called Hierarchical Mixture of Experts [79]. In these approaches a non-leaf class is represented by a *gating network* and a leaf class is represented by an *expert* *network*. Class predictions are made by the expert networks, and the gating networks determine which expert networks are allowed to make predictions. The method of Ruiz and Srinivasan [146] associates expert networks also with non-leaf classes and may therefore be able to predict non-leaf classes. However, the sparseness of the data and the trade-off is not a concern in this approach either. Standard performance measures are used in all of these studies. Moreover, only a tree is considered.

McCallum et al. [109] propose a quite different approach for naive Bayes classifiers. Their main concern is the sparseness of the data in the leaf classes. They therefore use a statistical technique called *Shrinkage*. This technique improves the probability estimates for the leaf classes by shrinking them towards the probability estimates for the superclasses. Very briefly, weights are associated with each class in the hierarchy. The class-conditioned word probabilities (i.e., $P(word|class)$) used in the naive Bayes algorithm are adjusted using these weights. This is done for each word and each leaf class such the probability of a word given the leaf class is a weighted sum of the probability of the class and the probabilities of superclasses. The weights are found by using a version of the EM-algorithm [36]. They are able to show a reduction in error up to 29%. Hence, this approach seems more interesting with regard to our concerns in the DAG. However, the approach assumes that the hierarchy is a tree and only one *leaf* class is predicted for each object. The approach is therefore incapable of dealing with the trade-off.

Hence, there exist some hierarchical text classification approaches. However, our learning and prediction algorithms are very different from these approaches and have a least two novel features. They handle a DAG, and they are able to trade off details for precision. As far as we know, there is no text classification approach that is able to perform such tasks. Moreover, most of the hierarchical text classification approaches predict only leaf classes and assume that the classes are labeled to leaf classes. So, our methods are also fairly novel in their ability to predict non-leaf classes. Only standard performance measures were used in these studies so that our evaluation method is also original.

Related work in rough sets

There are also some related approaches in rough set theory with regard to our framework. However, these relationships are more distant.

In our framework a partial order is defined on the classes. The dominance-based rough set approach (e.g., [62]) also considers an ordering of classes so that these problems may appear similar. However, there are important differences. In the dominance-based approach the ordering relation is a linear ordering (or at least a total preorder). Any two classes are therefore related since the relation is total. This means the membership of an object labeled to class c is not unknown with respect to subclasses of c since c does not have two or more unrelated subclasses. Hence, the unknown-property simply does not occur. Moreover, the inconsistency introduced by the DAG does not arise either.

The concept of unknown objects occurs in other settings in RST, however. Małuszyński and Vitoria [106] present an RST framework for Prolog with unknown objects. However, they develop a Rough Datalog language and define its semantics. They do not develop approximations and boundary sets for unknown objects. Learning classifiers is not a concern for them, either.

11.3 Further work

The work presented in this thesis may be continued in many different directions. This section describes some of these opportunities.

Classification of samples

The methodology developed for the classification of tumor samples seems to work quite well, but some improvements are possible. More work could be done at the feature selection stage. We have only tried one feature selection method, but there are many other methods that could have been used. Development of hypothesis testing methods for identifying differentially expressed genes in microarray data is currently a very active research area, and several methods have already been introduced (e.g., [179, 43, 38, 101]). Some of these may yield a better performance than the bootstrap method used here. This work could be continued by combining rough sets with other feature selection methods. The performance of these methods could then be compared. One may, of course, also try to develop new hypothesis testing methods, which later could be used with rough set methods.

Another opportunity for further research may be to identify combinations of genes that can be applied to build classifiers. With a combination we mean a set of genes where no gene possesses a discriminatory ability on its own, but the set as whole can discern between a set of classes. A combination corresponds basically to a reduct, and one could try to identify such combinations with the genetic reduct algorithm or another reduct algorithm. However, when the number of samples is small, most of the reducts found by such an algorithm will be artifacts of the data and will not have a real biological connection. Hence, a much larger data set would be required for this type of analysis than the gastric cancer data set. The exact number of samples that would be required has not be studied (as far as we are aware of). More work should be done on determining the necessary number of samples either by simulations or by analytical methods.

It may be possible in this case to reduce the required number of samples by filtering out irrelevant reducts. For example, one could split the data into a training set and a validation set. The training set could then be used for learning reducts, while the validation set could be used to test the quality of each reduct and to filter out the irrelevant reducts. Such a strategy could be combined with bootstrapping or cross-validation such that several training and validation sets were created. We could then count the number of times that a reduct would pass through filtering and select the most frequent ones (in a similar manner to the dynamic reduct approach).

In our study, the rough set methods were compared to discriminant analysis. There are obviously many learning methods that could be applied to classify tumors from microarray data. More work could thus be done on comparing rough set methods to other learning methods such as support vector machines and decision trees.

The evaluation of the classifiers could also be improved. We have computed the accuracy and the AUC of the classifiers and used these measures to compare different learning and discretization algorithms. However, the performance measures were computed with leave-one-out cross-validation on a relatively small set of samples such that we obtained only rough estimates of the performance. The results would have been stronger if the variations in these estimates had been taken into account when the classifiers were compared. Hence, it would be better to compute confidence intervals[1] for the AUC (and the accuracy) of each classifier and then compare the confidence intervals. In this case, it would be an advantage to have a larger set of tumors than the set that we had at our disposal. The width of a confidence interval depends on the number of tumors, and with only 17 samples the confidence intervals would probably be quite wide in most cases.

More work can also be done on the preprocessing of the data. It is sometimes difficult to decide which normalization and filtering method to use. The preprocessing could be improved by using direct references such as spiking controls. Such controls can be used in several ways. One possibility is to apply spiking controls for method selection and quality control. This can be done by measuring the differences between the observed and the actual ratios of the spiking controls. The filtering/normalization method that results in the smallest differences may then be selected.

However, spiking controls may be used more directly in the preprocessing process. We may, for example, place spiking controls with varying ratios and intensity levels in a regular pattern over the whole microarray surface and fit a normalization function to these controls. Such a normalization method will most likely be more accurate than the current normalization methods since the actual ratio and the actual intensity level of the spikes will be known. The method will therefore avoid the assumptions, which most normalization methods are based on. A potential problem, however, is that the spiking controls may behave differently from the rest of the spots since the cDNA for these controls are added to the samples. Experiments are thus required in order to determine whether this is an actual problem. This approach also introduces a cost since some of the spots have to be used for the spikes. However, most arrays have some empty spots that may be used for this purpose. So this cost may not be that large, and as the arrays become larger, it should diminish.

The analysis of the gastric tumors should be continued. The classifiers identified many connections between genes and clinical parameters. These connections should be verified experimentally.

The number of samples in the gastric data set was quite small, and several of the clinical parameters had very skewed class distributions. The number of genes was also

[1]In this case, we could compute a confidence interval of a performance measure directly or we could compute the standard error for the performance measure and then find the confidence intervals from the standard error.

relatively small compared to the size of the human genome. Thus, a much larger study with more genes and many more tumors should be performed. It is recommended that at least 50–100 microarrays are used in an experiment.

Classification of genes in an ontology

Our rough set framework for the DAG can be extended in several ways. Ziarko [197] has introduced a generalization of rough set theory, which is called the variable precision rough set (VPRS) model. In this model, the lower and the upper approximation are defined according to an arbitrary level $\pi \in [0, 0.5\rangle$.

$$\begin{aligned} \underline{B}_\pi X &= \{x \in U \mid \mu_B^X(x) \geq 1 - \pi\} \\ \overline{B}_\pi X &= \{x \in U \mid \mu_B^X(x) > \pi\} \end{aligned} \qquad \text{where} \quad \mu_B^X(x) = \frac{|[x]_B \cap X|}{|[x]_B|}$$

These approximations may sometimes give better results than the standard approximations. Thus, they should be introduced into our framework and be implemented in the learning algorithms.

The framework could also utilize the objects in the superclasses to a larger extent. When we consider a class c, we assume that objects of its superclasses may belong to c or any of its siblings. The class membership of these objects is therefore unknown. However, the boundary sets could try to determine whether these objects belong to c or any of siblings. We may, for example, assume that the objects of a superclass belong to the class with most similar objects and move the objects to this class. This assumption may be implemented by the means of proximity based boundary sets. These sets may generalize our boundary sets in much the same way as the VPRS approximations generalize the standard rough set approximations. However, in this case we will use a similarity measure such as $dist(I_1, I_2)$ from Section 8.5.2 rather than the rough membership function $\mu_B^X(x)$. An object x labeled to a superclass of c may, for example, be assigned to the known set of c if there is some object y labeled to c or one of the subclasses of c, and $1 - dist(Inf_A(x), Inf_A(y))/|A|$ is above some predetermined threshold.

Our framework is in some ways related Dempster-Shafer theory of evidence [154]. This theory can deal with missing or unknown information. It employs a belief function and a plausibility function that determine the certainty and the possibility that an object may belong to a set of classes. These functions correspond roughly to the known set and the potential set in our framework. It would thus be interesting to study the relationship between our framework and this theory. It may also be possible to apply Dempster-Shafer theory on the ontology. However, this theory does not allow both inconsistent and unknown objects. So some modifications would be necessary. The connection between rough set theory and Dempster-Shafer theory have previously been studied by Skowron and Grzymala-Busse [159]. Their work might be a good starting point for such research.

The bottom-up and ensemble methods may be developed further:

- **The bottom-up method**: The pruning system may be defined in many different ways. We have only tried one alternative, and it is possible that other pruning systems may work even better. Some alternate pruning strategies are:

  - **Pruning without thresholds**: In our approach the user must specify several pruning thresholds so that the algorithm can determine if a rule should be pruned. These thresholds might be avoided with other approaches. One possibility would be to use the classes at the top level as a yardstick. The performance of a rule learned for a detailed class could be compared to the performance of the rules made for the top level classes. If the performance was worse, the rule could be pruned. In this case, we would not need any thresholds. The algorithm would simply identify the most detailed classes with the highest achievable precision.
  - **Pruning without splitting**: The pruning system divides the original training data into a training sample and a validation sample in order to avoid overfitting. This means that this method needs more data than an approach that keeps the original data intact, and it may consequently lose a few details. It would be beneficial if this division could be avoided. This might be possible if a different pruning criterion is used. One possibility would be to apply the voting system of the ensemble method as pruning strategy. For example, we may compare the gain of a rule to the gain of the rules that have been created for the subclasses and prune the rules that have the least gain.

- **The ensemble method**: The voting system of the ensemble method was devised informally. A more formal approach could be conceived with a basis in Bayesian decision theory. A combination of the performance measures that were introduced in Chapter 9 could be used as a loss function (or as a gain function). A formal algorithm could then be derived from this loss function. Such an approach would allow us to investigate the properties of the voting system (e.g., whether the system is optimal or not).

The bottom-up and ensemble methods do not learn rules by themselves. Instead, they employ a separate algorithm, which learns the rules. They may therefore be regarded as higher level algorithms that apply a learning algorithm as a sub-algorithm. Two different sub-algorithms have been used in this thesis, namely the top-down and bottom-up search algorithms. They were chosen mainly for their simplicity and may not be the most accurate ones. Many other learning algorithms could be applied and may yield a better performance. Further work should explore the potential in combining the bottom-up and ensemble methods with other learning algorithms.

Any rule learning algorithm can be used directly by the bottom-up and ensemble methods. They just need to be implemented in our system. Some possible candidates are:

- The genetic reduct algorithm that was mentioned in Section 10.2.2. This may be less noise sensitive than the top-down and bottom-up methods. Hence, it may produce more detailed and accurate predictions.

- Beam search algorithms may give a better performance.
- Bagging and boosting the top-down and bottom-up methods may also improve the precision and the detail level.

However, our methods are not limited to rule learning approaches. The bottom-up method can be applied to any binary classifier. We will just prune whole classes instead of rules in this case. More precisely, the algorithm will learn a binary classifier from the training sample when a class is visited. This binary classifier will then be tested on the validation sample and may be pruned if its performance is unsatisfactory.

The ensemble method can also be adapted to other learning approaches, but this conversion may be slightly more complicated. The consenting and the dissenting rules of a class constitute a binary classifier. This means that most learning algorithms may in principle be applied since most algorithms can learn such a classifier. However, the voting system requires several different kinds of votes, and the SC support and the SN support must at least be estimated (The SA support and the WB support can be estimated from the SC support). Some methods may be incapable of providing such estimates directly. These methods may not be applicable without modifications. Many learning methods, nevertheless, provide a measure that assesses their confidence of a class prediction. One possibility would be to train two binary classifiers — one for the SC support and one for the SN support. The SC support and the SN support could then be estimated from the confidence measure of the respective classifier.

Such a strategy should be very easy to implement with a Bayesian learning approach. The consenting votes are basically an estimate of the probability that the class should be predicted, and the dissenting votes are an assessment of the probability that the class should not be predicted. Bayesian learning approaches like the naive Bayes classifier and Bayesian networks output such probabilities. They may thus be applied readily with our methods. Note that Bayesian learning algorithms are often used for text classification problems and could be applied to predict gene function from Medline abstracts (see below). Hence, they may be a good starting point for further work.

More research may be done on the performance measures. In our measures, the maximal relative depth is used to assess how well a prediction reassembles an annotation. However, many other functions could be used for this purpose. Some alternatives are the penalty functions that were introduced in Section 8.3.2. A further study should identify such functions and determine which gives the best and most interpretable assessments.

The fibroblast data set was quite small and had a limited number of patterns (cf. the clustering of Iyer et al.). Hence, it was not ideal for prediction of gene function. The bottom-up and ensemble methods should therefore be applied to much larger data sets. One possibility would be to use microarray data from several experimental studies assuming that these contain a larger variety of expression patterns such that more classes can be separated. However, it is a question whether enough microarray data can be obtained so that both accurate and detailed predictions can be made. As mentioned in Section 3.5.1 on page 38, the assumption that co-expressed genes are involved in the same biological process has at least been contested by Shatkay et al. [156]. So, it is possible that gene function cannot be predicted with high accuracy from gene expression data alone.

Still, it is feasible to combine microarray data with other kinds of data. Gene function may also be predicted from:

- DNA sequence data, and
- Biomedical documents such as Medline abstracts

We could try to combine microarray data with one of these kinds of data or with both simultaneously. One possibility would be to add information about transcription factor binding sites in the promoter region to a microarray data set. This could be done by creating Boolean attributes that denoted the presence or absence of a particular binding site in a gene. One may also try to predict the gene function from only Medline abstracts or sequence data.

An important feature of our methods is their generality. They can be applied on many different kinds of data and are not limited to microarray data and gene function prediction. One example is text classification. Further research should explore the utility of our methods on such problems.

The fibroblast data were discretized with the template approach of Hvidsten et al. [75]. This approach produces very general patterns from the data and may lose a lot of information. Other discretization methods may give better results. Thus, different representation forms should be tried with the fibroblast data and other microarray data.

Even though it is beyond the scope of this thesis, it is evident that the Gene Ontology needs cleansing. It is still very immature, and the semantics provided by the GO Consortium is informal and possibly ill-conceived. As we have pointed out, there are conflicts between the semantics and the actual ontologies. These conflicts may reduce classification performance. Thus, they should be resolved, and a clear and formal semantics should be provided.

Hence, there are still many opportunities for further research. We believe that such work may prove to be prosperous in the future.

Some Concepts from Set and Graph Theory

A

A.1 Introduction

Several concepts from set and graph theory are used in Chapters 6 and 7. Some of these concects are repeated here. However, it is *not* the intention to give a complete review of these concepts.

A.2 Sets

A set is a collection of objects. It can be specified by listing the objects inside curly brackets $\{\}$. For example, $\{a, b, c\}$ is a set that consists of the objects a, b, and c. An object is said to be a member of a set if it occurs in this set. This is denoted with the symbol $\in$ such that $a \in A$ states that the object a is a member of the set A.

Two sets play special roles in set theory. The set of all possible objects is called the universe of discourse and is denoted as U. The set that does not contain any objects is called the empty set and is denoted as $\emptyset$.

The following operators are usually defined on sets.

Definition A.1 (Subset). Let A and B be sets taken from the universe U. Then A is a subset of B, denoted as $A \subseteq B$, iff $x \in A$ implies $x \in B$. ■

In the following we assume that universe U is equal to $\{a, b, c, d, e, f\}$, and that $X = \{a, b, c, d\}$, $Y = \{c, d\}$, and $Z = \{e, d\}$. In this case, $X \subseteq X$, $Y \subseteq X$, and $Z \nsubseteq X$ are true.

Definition A.2 (Proper Subset). Let A and B be sets taken from the universe U. Then A is a subset of B, denoted as $A \subseteq B$, iff $A \subseteq B$ and $A \neq B$. ■

For example, $X \not\subset X$, $Y \subset X$, and $Z \not\subset X$ are true.

Definition A.3 (Union). Let A and B be sets taken from the universe U. The union of A and B is denoted as $A \cup B$ and is equal to $\{x \in U \mid x \in A \text{ or } x \in B\}$. ■

For example, $X \cup Y = \{a, b, c, d\}$, $X \cup Z = \{a, b, c, d, e\}$, and $Y \cup Z = \{c, d, e\}$.

Definition A.4 (Intersection). Let A and B be sets taken from the universe U. The intersection of A and B is denoted as $A \cap B$ and is equal to $\{x \in U \mid x \in A \text{ and } x \in B\}$. ■

For example, $X \cap Y = \{c, d\}$, $X \cap Z = \{d\}$, and $Y \cap Z = \{d\}$.

Definition A.5 (Complement). Let A be a set taken from the universe U. The complement of A is denoted as $U - A$ and is equal to $\{x \in U \mid x \notin A\}$. ■

For example, $U - X = \{e, f\}$, $U - Y = \{a, b, e, f\}$, and $U - Z = \{a, b, c, f\}$.

Definition A.6 (Difference). Let A and B be sets taken from the universe U. The difference $A - B$ denotes the objects that are members of A, but not of B, i.e., $A - B = \{x \in U \mid x \in A \text{ and } x \notin B\}$. ■

For example, $X - Y = \{a, b\}$, $X - Z = \{a, b, c\}$, and $Y - Z = \{c\}$.

Definition A.7 (Symmetric difference). Let A and B be sets taken from the universe U. The symmetric difference $A \triangle B$ denotes the objects that are members of either A or B, i.e., $A \triangle B = (A - B) \cup (B - A)$. ■

For example, $X \triangle Y = \{a, b\}$, $X \triangle Z = \{a, b, c, e\}$, and $Y \triangle Z = \{c, e\}$.

Definition A.8 (Power set). Let A be a set taken from the universe U. The power set is denoted as $\mathcal{P}(A)$ and contains all subset of A, i.e., $\mathcal{P}(A) = \{B \subseteq U \mid B \subseteq A\}$ ■

For example, $\mathcal{P}(\{a, b, c\}) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$.

Many different properties can be derived from these definitions. The most important ones are given in the following theorem.

Theorem A.1. *Let A, B, and C be sets taken from the universe U. Then the following properties hold.*

- ***Law of double complement:*** $A = (U - (U - A))$
- ***Idempotent laws:*** $A \cup A = A$
  $A \cap A = A$
- ***Identity laws:*** $A \cup \emptyset = A$
  $A \cap U = A$
- ***Inverse laws:*** $A \cup (U - A) = U$
  $A \cap (U - A) = \emptyset$
- ***Domination laws:*** $A \cup U = U$
  $A \cap \emptyset = \emptyset$

- ***Commutative laws:*** $A \cup B = B \cup A$
  $A \cap B = B \cap A$
- ***Associate laws:*** $A \cup (B \cup C) = (A \cup B) \cup C$
  $A \cap (B \cap C) = (A \cap B) \cap C$
- ***Distributive laws:*** $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
  $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
- ***DeMorgan's laws:*** $U - (A \cup B) = (U - A) \cap (U - B)$
  $U - (A \cap B) = (U - A) \cup (U - B)$
- ***Absorption laws:*** $A \cup (A \cap B) = A$
  $A \cap (A \cup B) = B$ ■

Some special properties are used in Chapter 7. These are proven in the next lemmas.

Lemma A.1. *Given, $B \subseteq C$, it is true that*

a) $A \cap C = \emptyset$ *implies* $A \cap B = \emptyset$

b) $A \cap B \neq \emptyset$ *implies* $A \cap C \neq \emptyset$

Proof. Assume that the first option is false. Then, $A \cap B \neq \emptyset$ and $A \cap C = \emptyset$ must be true. There must be some $x \in A \cap B$, and $x \in A$ and $x \in B$ must hold as well. However, $x \notin A \cap C$ since $A \cap C = \emptyset$. This means that $x \notin C$ (since $x \in A$), and B is consequently not a subset of C. Contradiction. So the first option is true. The second option is equivalent to the first (Just negate the implication). □

Lemma A.2.

a) $A \cap B = \emptyset$ *and* $A \cap C = \emptyset$ *iff* $A \cap (B \cup C) = \emptyset$

b) $A \cap B \neq \emptyset$ *or* $A \cap C \neq \emptyset$ *iff* $A \cap (B \cup C) \neq \emptyset$

Proof.

a) $A \cap B = \emptyset$ and $A \cap C = \emptyset \Leftrightarrow (A \cap B) \cup (A \cap C) = \emptyset \Leftrightarrow A \cap (B \cup C) = \emptyset$.

b) This is equivalent to the first option (Just negate both sides of the iff). □

Lemma A.3.

a) $A \cap B = \emptyset$ *or* $A \cap C = \emptyset$ *implies* $A \cap (B \cap C) = \emptyset$

b) $A \cap (B \cap C) \neq \emptyset$ *implies* $A \cap B \neq \emptyset$ *and* $A \cap C \neq \emptyset$ ■

Proof.

a) This is equivalent to the second option.

b) Since $A \cap (B \cap C) \neq \emptyset$, there must be some $y \in A \cap (B \cap C)$. Then $y \in A$, $y \in B$, and $y \in C$. This means that $A \cap B \neq \emptyset$ and $A \cap C \neq \emptyset$. □

Note that the implication in the other direction does not hold. Consider the first option. If B and C are disjoint subsets of A (i.e., $B \cap C = \emptyset$), then $A \cap (B \cap C) = \emptyset$ is true, but neither $A \cap B = \emptyset$ nor $A \cap C = \emptyset$ is true.

Lemma A.4. *For any sets A and B, it holds that $A \cap B \subseteq A$ and $A \cap B \subseteq B$. Moreover, $A \subseteq A \cup B$ and $B \subseteq A \cup B$.* ■

Proof. Assume that $A \cap B \subseteq A$ is false. Then there is an x in $A \cap B$, but $x \notin A$. However, $x \in A \cap B$ implies that $x \in A$. So this is impossible, and $A \cap B \subseteq A$ must be true. The proof of $A \cap B \subseteq B$ is similar.

Assume that $A \subseteq A \cup B$ is false. Then there is an x in A, but not in $A \cup B$. However, $x \in A \cup B$ must be true since $x \in A$. So this leads to a contradiction, and $A \subseteq A \cup B$ must hold. The proof of $B \subseteq A \cup B$ is similar. □

Lemma A.5. *Let A, B, and C be sets where $A \subseteq C$ and $B \subseteq C$. Then $A \cup B \subseteq C$, and $A \cap B \subseteq C$ hold.* ■

Proof. Assume that $A \cup B \subseteq C$ does not hold. Then there is an $x \in A \cup B$ and x is not in C. Since $x \in A \cup B$, $x \in A$ or $x \in B$ must be true. However, $A \not\subseteq C$ if $x \in A$ and $B \not\subseteq C$ if $x \in B$. This means that x cannot be in $A \cup B$. Hence, $A \cup B \subseteq C$ must hold. The proof of $A \cap B \subseteq C$ is similar. □

A.3 Relations

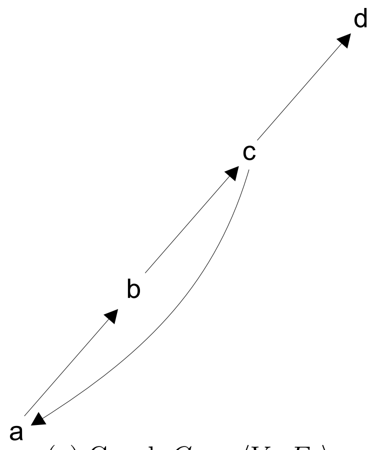
Definition A.9. A cartesian product of A and B, denoted $A \times B$, is defined as $\{\langle a, b\rangle \mid a \in A, b \in B\}$. More generally, the cartesian product $A_1 \times A_2 \times \cdots \times A_n$ equals $\{\langle a_1, a_2, \ldots, a_n\rangle \mid a_1 \in A_1, a_2 \in A_2, \ldots, a_n \in A_n\}$ ■

For example, let $A = \{1, 2\}$ and $B = \{3, 4\}$. Then we have that $A \times B = \{\langle 1, 3\rangle, \langle 1, 4\rangle, \langle 2, 3\rangle, \langle 2, 4\rangle\}$.

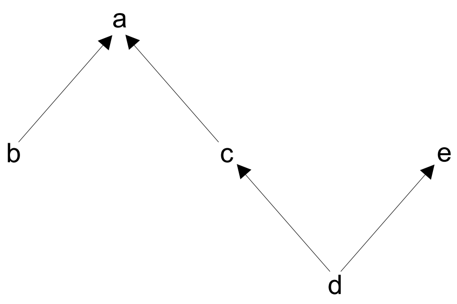
Definition A.10. A relation R is subset of a cartisan product. ■

Definition A.11. A binary relation R on X is:

- **Reflexive**: If xRx for all $x \in X$
- **Symmetric**: If xRy implies yRx for all $x, y \in X$
- **Anti-symmetric**: If xRy and yRx imply $x = y$ for all $x, y \in X$



(a) Graph $G_1 = \langle V_1, E_1 \rangle$



(b) Graph $G_2 = \langle V_2, E_2 \rangle$

Figure A.1: *Examples of graphs.*

- **Transitive**: If xRy and yRz imply xRz for all $x, y, z \in X$
- **Total**: If xRy or yRx holds for all $x, y \in X$

R is an equivalence relation if it is reflexive, symmetric, and transive. R forms a partial order if it is reflexive, antisymmetric, and transitive. ■

A.4 Graphs

Definition A.12 (Graph). A *directed graph* G is a tuple $\langle V, E \rangle$ where V is a set of nodes (or vertices) and $E \subseteq V \times V$ is a set of edges between nodes. ■

One example of a directed graph is $G_1 = \langle V_1, E_1 \rangle$ where $V_1 = \{a, b, c, d\}$ and $E_1 = \{\langle a, b \rangle, \langle b, c \rangle, \langle c, a \rangle, \langle c, d \rangle\}$.

Definition A.13 (Path & Cycles). A *path* from node v_0 to node v_n in graph $G = \langle V, E \rangle$ is a sequence of nodes $\langle v_0, v_1, \ldots, v_n \rangle$ such that $\langle v_{i-1}, v_i \rangle \in E$ for all $i = 1, \ldots, n$. A path forms a *cycle* if $v_0 = v_n$. ■

The graph G_1 has a path $\langle a, b, c, d \rangle$ from a to d. It has also a path $\langle a, b, c, a \rangle$, which is a cycle.

Definition A.14 (DAG). A directed acyclic graph (DAG) is a directed graph $G = \langle V, E \rangle$ with no cycles. ■

The graph G_1 has a cycle. Hence, it is not a DAG. However, the graph $G_2 = \langle V_2, E_2 \rangle$ where $V_2 = \{a, b, c, d, e\}$ and $E_2 = \{\langle b, a \rangle, \langle c, a \rangle, \langle d, c \rangle, \langle d, e \rangle\}$ is a DAG.

Definition A.15 (Root). Let $G = \langle V, E \rangle$ be a DAG. $r \in V$ is a *root* if there is no egde from r to another node, i.e., $\langle r, v \rangle \notin E$ holds for all $v \in V$. ■

The graph G_2 has two roots a and e. There are no edges from these nodes to another node.

Definition A.16 (Rooted DAG). A rooted DAG $G = \langle V, \top, E \rangle$ is a DAG where $\top$ is a root and for all $v \in (V - \{\top\})$ there is a path from v to $\top$. ■

G_2 is not rooted since it has two roots. There is no path from e to a, and vice versa. However, the graph would be a rooted DAG if the node e and the edge $\langle d, e \rangle$ were removed. In this case, a would be the root.

Bibliography

[1] Kamal M. Ali and Michael J. Pazzani. HYDRA: A noise-tolerant relational concept learning algorithm. In R. Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 1064–1071, 1993.

[2] Ash A. Alizadeh, Michael B. Eisen, R. Eric Davis, Chi Ma, Izidore S. Lossos, Andreas Rosenwald, Jennifer C. Boldrick, Hajeer Sabet, Truc Tran, Xin Yu, John I. Powell, Liming Yang, Gerald E. Marti, Troy Moore, James Hudson Jr., Lisheng Lu, David B. Lewis, Robert Tibshirani, Gavin Sherlock, Wing C. Chan, Timothy C. Greiner, Dennis D. Weisenburger, James O. Armitage, Roger Warnke, Ronald Levy, Wyndham Wilson, Michael R. Grever, John C. Byrd, David Botstein, Patrick O. Brown, and Louis M. Staudt. Distinct types of diffuse large B-cell lymphoma identified by gene expression profiling. *Nature*, 403(6769):503–511, 3 February 2000.

[3] Claus L. Andersen, Outi Monni, Urs Wagner, Juha Kononen, Maarit Bärlund, Christoph Bucher, Antonio Haas, Phillipe amd Nocito, Heidi Bissig, Guido Sauter, and Anne Kallioniemi. High-throughput copy number analysis of 17q23 in 3520 tissue specimens by fluorescence in situ hybridization to tissue microarrays. *American Journal of Pathology*, 161(1):73–79, 2002.

[4] F. Baader and W. Nutt. Basic description logics. In F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors, *The Description Logic Handbook*, pages 47–100. Cambridge University Press, 2002. In press.

[5] Richardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval.* ACM Press, 1999.

[6] Trent Basarsky, Damian Verdnik, Jack Ye Zhai, and David Wellis. Overview of a microarray scanner: Design essential for an integrated acquisition and analysis platform. In Schena [150], pages 265–284.

[7] Jan G. Bazan. Dynamic reducts and statistical inference. In *Proceedings of the 6th International Conference on Information Processing and Management of*

Uncertainty in Knowledge-Based Systems (IPMIU'96), volume 3, pages 1147–1152, 1996.

[8] Jan G. Bazan. A comparison of dynamic and non-dynamic rough set methods for extracting laws from decision table. In Polkowski and Skowron [138], pages 321–365.

[9] T. Beißbarth, K. Fellenberg, B. Brors, R. Arribas-Prat, J. M. Boer, N. C. Hauser, M. Scheideler, J. D. Hoheisel, G. Schütz, A. Poustka, and M. Vingron. Processing and quality control of DNA array hybridization data. *Bioinformatics*, 16(11):1014–1022, 2000.

[10] Amir Ben-Dor, Laurakey Bruhn, Nir Friedman, Iftach Nachman, Michèl Schummer, and Zohar Yakhini. Tissue classification with gene expression profiles. In Shamir et al. [155], pages 54–64.

[11] Axel Bernal, Uy Ear, and Nikos Kyrpides. Genomes online database (GOLD): A monitor of genome projects world-wide. *Nucleic Acids Research*, 29(1):126–127, 2001. **http://ergo.integratedgenomics.com/GOLD/**.

[12] M. Bittner, P. Meltzer, Y. Chen, Y. Jiang, E. Seftor, M. Hendrix, M. Radmacher, R. Simon, Z. Yakhini, A. Ben-Dor, N. Sampas, E. Dougherty, E. Wang, F. Marincola, C. Gooden, J. Lueders, A. Glatfelter, P. Pollock, J. Carpten, E. Gillanders, D. Leja, K. Dietrich, C. Beaudry, M. Berens, D. Alberts, and V. Sondak. Molecular classification of cutaneous malignant melanoma by gene expression profiling. *Nature 2000*, 406(6795):536–540, Aut 3 2000.

[13] M. L. Bittner, Y. Chen, A. N. Dorsel, and E. R. Dougherty, editors. *Microarrays: Optical Technologies and Informatics*, volume 4266 of *Proceedings of SPIE, Society of Optical Engineering*, San Jose, California, USA, January 2001.

[14] A.P. Blanchard, R.J. Kaiser, and L.E. Hood. High-density oligonucleotide arrays. *Biosensors & Bioelectronics*, 11(6/7):687–690, 1996.

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

[16] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification And Regression Trees*. Waldsworth & Brooks/Cole, 1984.

[17] Sydney Brenner, Maria Johnsen, John Bridgham, George Golda, David H. Lloyd, Davida Johnson, Shujun Lou, Sarah McCurdy, Michael Foy, Mark Ewan, Rithy Roth, Dave George, Sam Electr, Glenn Albrecht, Eric Vermaas, Steven R. Williams, Keith Moon, Timothy Burcham, Michael Pallas, Robert DuBridge, James Kirchner, Karen Fearon, Jen-I Mao, and Kevin Corcoran. Gene expression analysis by massively parallel signature sequencing (MPSS) on microbead arrays. *Nature Biotechnology*, 18(6):630–634, 2000.

[18] Sydney Brenner, Steven R. Williams, Eric H. Vermaas, Thorsten Storck, Keith Moon, Christie McCollum, Jen-I Mao, Shujun Luo, James J. Kirchner, Sam Eletr, Robert B. DuBridge, Timothy Burcham, and Gleen Albrecht. In vitro cloning of complex mixtures of DNA on microbeads: Physical separation of differentially expressed cDNAs. *Proceedings of the National Academy of Sciences, USA*, 97(4):1665–1670, 2000.

[19] Michael P. S. Brown, William Noble Grudy, David Lin, Nello Cristianini, Charles W. Sugnet, Terrence S. Furey, Manuel Ares, Jr., and David Haussler. Support vector machine classification of microarray gene expression data. Technical Report USCS-CRL-99-09, University of California, 1999.

[20] Michael P. S. Brown, William Noble Grundy, David Lin, Nello Cristianini, Charles W. Sugnet, Terrence S. Furey, Manuel Ares, Jr., and David Haussler. Knowledge-based analysis of microarray gene expression data by using support vector machines. *Proceedings of the National Academy of Sciences, USA*, 97(1):262–267, 2000.

[21] Patrick O. Brown and Davied Botstein. Exploring the new world of genome with DNA microarrays. *Nature Genetics*, 21:33–37, January 1999. Supplement.

[22] Søren Brunak and Anders Krogh, editors. *Proceedings for the 9th International Conference on Intelligent Systems for Molecular Biology (ISMB-2001)*, Bioinformatics 17 (6), 2001.

[23] Soumen Chakrabarti, Byron Dom, Rakesh Agrawal, and Prabhakar Raghavan. Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. *VLDB Journal*, 7(3):163–178, 1998.

[24] Yidong Chen, Edward R. Dougherty, and Michael L. Bittner. Ratio-based decision and the quantitative analysis of cDNA microarray images. *Journal of Biomedical Optics*, 2(4):364–374, 1997.

[25] Raymond J. Cho, Mingxia Huang, Michael J. Campbell, Helin Dong, Lars Steinmetz, Lisa Sapinoso, Garret Hampton, Stephen J. Elledge, Ronald W. Davis, and David J. Lockhart. Transcriptional regulation and function during the human cell cycle. *Nature Genetics*, 27(1):48–54, 2001.

[26] S. Chu, J. DeRisi, M. Eisen, J. Mulholland, D. Botstein, P.O. Brown, and I. Herskowitz. The transcriptional program of sporulation in budding yeast. *Science*, 282(5389):699–705, 23. October 1998.

[27] W. S. Cleveland and C. L. Loader. Smoothing by local regression: Principles and methods. In W. Härdle and M. G. Schimek, editors, *Statistical Theory and Computational Aspects of Smoothing*. Springer, 1996.

[28] Margaret H. Coletti and Howard L. Bleich. Medical subject headings used to search the biomedical literature. *Journal of the American Medical Informatics Association*, 8(4):317–323, 2001.

[29] The Gene Ontology Consortium. Gene Ontology: Tool for the unification of biology. *Nature Genetics*, 25(1):25–29, 2000.

[30] The Gene Ontology Consortium. Creating the Gene Ontology resource: Design and implementation. *Genome Research*, 11:1425–1433, 2001.

[31] The Gene Ontology Consortium. Gene Ontology usage guide, revision: 1.26, July 7, 2002. **http://www.geneontology.org/doc/GO.usage.html**.

[32] The International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):4387–4391, 15. February 2001.

[33] Nello Cristianini and Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, Cambridge, UK, 2000.

[34] M. Dash and H. Liu. Feature selection for classification. *Intelligent Data Analysis*, 1:131–156, 1997.

[35] Amaryllis Deliyanni and Robert A. Kowalski. Logic and semantic networks. *Communications of the ACM*, 22(3):184–192, March 1979.

[36] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, B 39:1–38, 1977.

[37] Radhika Desikan, Soheila A.-H.-Mackerness, John T. Hancock, and Steven J. Neill. Regulation of the arabidopsis transcriptome by oxidative stress. *Plant Physiology*, 127(1):156–172, 2001.

[38] Sandrine Dudiot, Yee Hwa Yang, Matthew J. Callow, and Terence P. Speed. Statistical methods for identifying differentially expressed genes in replicated cDNA microarray experiments. *Statistica Sinica*, 12(1):111–140, 2002.

[39] Sandrine Dudoit, Jane Fridlyand, and Terence P. Speed. Comparison of discrimination methods for the classification of tumors using gene expression data. Technical Report 567, Dept. of Statistics, University of California, Berkeley, June 2000. **http://www.stat.berkeley.edu/users/terry/zarray/Html/discr.html**.

[40] David J. Duggan, Michael Bittner, Yidong Chen, Paul Meltzer, and Jeffrey M. Trent. Expression profiling using cDNA microarrays. *Nature Genetics*, 21:10–14, January 1999. Supplement.

[41] Susan T. Dumais and Hao Chen. Hierarchical classification of web content. In Nicholas J. Belkin, Peter Ingwersen, and Mun-Kew Leong, editors, *Proceedings of the 23rd ACM International Conference on Research and Development in Information Retrieval (SIGIR-00)*, pages 256–263, Athens, GR, 2000. ACM Press, New York, US.

[42] Selina S. Dwight, Midori A. Harris, Kara Dolinski, Catherine Ball, Gail Binkely, Karen R. Christie, Dianna G. Fisk, Laurie Issel-Tarver, Mark Shroeder, Gavin Sherlock, Anand Sethuraman, Shuai Weng, David Botstein, and J. Michael Cherry. *Saccharomyces* Genome Database provides secondary gene annotation using the Gene Ontology. *Nucleic Acids Research*, 30(1):69–72, 2002.

[43] Bradley Efron, Robert Tibshirani, John D. Storey, and Virginia Tusher. Empirial bayes analysis of a microarray experiment. *Journal of the American Statistical Association*, 96:1151–1160, 2001.

[44] Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*, volume 57 of *Monographs on Statistics and Applied Probability.* Chapman & Hall, 1993.

[45] J.P. Egan. *Signal Detection Theory and ROC Analysis.* Academic Press, 1975.

[46] Bodo Eickhoff, Bernhard Korn, Matthias Schick, Annemarie Poustka, and Juergen van der Bosch. Normalization of array hybridization experiments in differential gene expression analysis. *Nucleic Acids Research*, 27(22):e33, 1999.

[47] Michael Eisen. *ScanAlyze User Manual.* Stanford University, 1999. **http://rana.lbl.gov/EisenSoftware.htm**.

[48] Michael B. Eisen, Paul T. Spellman, Patrick O. Brown, and David Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences, USA*, 95(25):14863–14868, 1998.

[49] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In R. Bajcsy, editor, *Proceedings of the 13th International Joint Conference of Artificial Intelligence (IJCAI-93)*, pages 1022–1027. Morgan Kaufmann, 1993.

[50] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery: An overview. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 1. AAAI Press, 1996.

[51] D. Fensel, I. Horrocks, F. van Harmelen, S. M. Decker, M. Erdmann, and M. Klein. OIL in a nutshell. In Rose Dieng and Olivier Corby, editors, *Proceedings of the 12th International Conference on Knowledge Engineering and Knowledge Management (EKAW-2000)*, number 1937 in Lecture Notes in Artificial Intelligence, pages 1–16. Springer-Verlag, 2000.

[52] David Finkelstein, Rob Ewing, Jeremy Gollub, Fredrik Sterky, and J. Michael Cherry. Microarray data quality analysis: Lessons from AFGC project. *Plant Molecular Biology*, 48(1/2):119–131, 2002.

[53] David B. Finkelstein, Jeremy Gollub, and J. Michael Cherry. Normalization and systematic measurement error in cDNA microarray data. In *Joint Statistical Meeting*, 2000. **http://afgc.stanford.edu/afgc_html/site2Stat.htm**.

[54] David B. Finkelstein, Jeremy Gollub, Rob Ewing, Fredrik Sterky, Shauna Somerville, and J. Michael Cherry. Iterative linear regression by sector: renormalization of cDNA microarray data and cluster analysis weighted by cross homology. In Lin and Johnson [93], pages 57–67.

[55] Stephen P.A. Fodor, Richard P. Rava, Xiaohua C. Huang, Ann C. Pease, Christopher P. Holmes, and Cynthia L. Adams. Multiplexed biochemical assays with biological chips. *Nature*, 364:555–556, 1993.

[56] Stephen P.A. Fodor, J. Leighton Read, Michael C. Pirrung, Lubert Stryer, Amy Tsai Lu, and Dennis Solas. Light-directed, spatially addressable parallel chemical synthesis. *Science*, 251:767–773, 1991.

[57] Dmitrij Frishman, Kaj Albermann, Jean Hani, Klaus Heumann, Agnes Metanomski, Alfred Zollner, and Hans-Werner Mewes. Functional and structural genomics using PEDANT. *Bioinformatics*, 17(1):44–57, 2001.

[58] Terrence S. Furey, Nello Cristianini, Nigel Duffy, David W. Bednarski, Michèl Schummer, and David Haussler. Support vector machine classification and validation of cancer tissue samples using microarray expression data. *Bioinformatics*, 16(10):906–914, October 2000.

[59] Johannes Fürnkranz. Separate-and-conquer rule learning. *Artificial Intelligence Review*, 13(1):3–54, 1999. Occurs also as Technical Report OEFAI-TR-96-25.

[60] T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesiro, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E.S. Lander. Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science*, 286:531–536, 1999.

[61] Z. Gou, R.A. Guilfoyle, A.J. Thiel, R. Wang, and L.M. Smith. Direct fluorescence analysis of genetic polymorphisms by hybridization with oligonucleotide arrays on glass supports. *Nucleic Acids Research*, 22(24):5456–5465, 1994.

[62] Salvatore Greco, Bendetto Matarazzo, and Roman Slowinski. A new rough set approach to multicriteria and multiattribute classification. In *Rough Sets and Current Trends in Computing*, volume 1424 of *Lecture Notes in Artificial Intelligence*, pages 60–67. Springer-Verlag, 1998.

[63] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[64] Jerzy W. Grzymala-Busse. Knowledge acquisition under uncertainty – a rough set approach. *Journal of Intelligent and Robotic System*, 1:3–16, 1988.

[65] Jerzy W. Grzymala-Busse. LERS – A system for learning from examples based on rough sets. In *Intelligent decision support: Handbook of Applications and Advances of Rough Sets Theory* [163], pages 3–18.

[66] Joseph G. Hacia. Resequencing and mutational analysis using oligonucleotide microarrays. *Nature Genetics*, 21:42–47, January 1999. Supplement.

[67] Christina A. Harrington, Carsten Rosenow, and Jacques Retief. Monitoring gene expression using DNA microarrays. *Current Opinions in Microbiology*, 3(3):285–291, 2000.

[68] Alexander J. Hartemink, David K. Gifford, Tommi Jaakkola, and Richard A. Young. Maximum likelihood estimation of optimal scaling factors for expression array normalization. In Bittner et al. [13].

[69] Trevor Hastie, Robert Tibshirani, David Botstein, and Patrick Brown. Supervised harvesting of expression trees. *Genome Biology*, 2(1):research0003.1–research0003.12, 2001.

[70] Trevor Hastie, Robert Tibshirani, Michael Eisen, Patrick Brown, Ross Doug, Uwe Scherf, John Weinstein, Ash Alizadeh, Louis Staudt, and David Botstein. Gene shaving: A new class of clustering methods for expression arrays. Technical report, Dept. of Statistics, Stanford, 2000. **http://www-stat.stanford.edu/~tibs/research.html**.

[71] Trevor Hastie, Robert Tibshirani, Michael B. Eisen, Ash Alizadeh, Ronald Levy, Louis Staudt, Wing C. Chan, David Botstein, and Patrick Brown. 'Gene shaving' as a method for identifying distinct sets of genes with similar expression patterns. *Genome Biology*, 1(2):research0003.1–research0003.21, 200.

[72] Robert C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11(1):63–91, 1993.

[73] Neal S. Holter, Madhusmita Mitra, Amos Maritan, Marek Cieplak, Jayanth R. Banavar, and Nina V. Fedoroff. Fundamental patterns underlying gene expression profiles: Simplicity for complexity. *Proceedings of the National Academy of Sciences, USA*, 97(15):8409–8414, 2000.

[74] Timothy Hughes, Mao Mao, Allan R. Jones, Julja Burhard, Matthew J. Marton, Karen W. Shannon, Steven M. Lefkowitz, Michael Ziman, Janell M. Schelter, Michael R. Meyer, Sumire Kobayashi, Colleen Davis, Hongyue Dai, Yudong D. He, Sergey B. Stephaniants, Guy Cavet, Wynn L. Walker, Anne West, Ernest

Coffey, Daniel D. Shoemaker, Roland Stoughton, Alan P. Blanchard, Stephen H. Friend, and Peter S. Linsley. Expression profiling using microarrays fabricated by an ink-jet oligonucleotide synthesizer. *Nature Biotechnology*, 19:342–347, April 2001.

[75] Torgeir R. Hvidsten, Jan Komorowski, Arne K. Sandvik, and Astrid Lægreid. Predicting gene function from gene expressions and ontologies. In Russ B. Altman, A. Keith Dunker, Lawrence Hunter, Kevin Lauderdale, and Teri E. Klein, editors, *Proceedings of the Pacific Symposium on Biocomputing 6 (PSB-2001)*, pages 299–310. World Scientific Press, 2001.

[76] Wishwanath R. Iyer, Michael B. Eisen, Douglas T. Ross, Greg Schuler, Troy Moore, Jeffrey C.F. Lee, Jeffrey M. Trent, Louis M. Staudt, James Hudson Jr., Mark S. Boguski, Deval Lashkari, Dari Shalon, David Botstein, and Patrick O. Brown. The transcriptional program in the response of human fibroblasts to serum. *Science*, 283:83–87, 1999.

[77] David S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[78] Richard A. Johnson and Dean W. Wichern. *Applied Multivariate Statistical Analysis*. Prentice-Hall, 1999.

[79] Michael I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6:181–214, 1994.

[80] Michael D. Kane, Timothy A. Jatkoe, Craig R. Stumpf, Jia Lu, Jeffrey D. Thomas, and Steven J. Madore. Assessment of the sensitivity and specificity of oligonucleotide (50mer) microarrays. *Nucleic Acids Research*, 28(22):4552–4557, 2000.

[81] Peter D. Karp. An ontology for biological function based on molecular interactions. *Bioinformatics*, 16(3):269–285, 2000.

[82] Andrew D. Keller, Michel Schummer, Lee Hood, and Walter L. Ruzzo. Bayesian classification of DNA array expression data. Technical Report UW-CSE-2000-08-01, Dept. of Computer Science & Engineering, University of Washington, August 2000.

[83] Thomas B. Kepler, Lynn Crosby, and Kevin T. Morgan. Normalization and analysis of DNA microarray data by self-consistency and local regression. Working Paper 00-09-055, Santa Fe Institute, 2000. **http://www.santafe.edu/sfi/publications/Abstracts/00-09-055abs.html**.

[84] M. Kathleen Kerr, Cynthia A. Afshari, Lee Bennett, Pierre Bushel, Jeanelle Martinez, Nigel J. Walker, and Gary A. Churchill. Statistical analysis of a gene expression microarray experiment with replication. *Statistica Sinica*, 12(1):203–217, 2002.

[85] M. Kathleen Kerr, Mitchell Martin, and Gary A. Churchill. Analysis of variance for gene expression microarray data. *Journal of Computational Biology*, 7(6):819–837, 2000.

[86] Tuevo Kohonen. Self-organization formation of topolocially correct feature maps. *Biological Cybernetics*, 43(1):59–69, 1982.

[87] Daphne Koller and Mehran Sahami. Hierarchically classifying documents using very few words. In Douglas H. Fisher, editor, *Proceedings of the 14th International Conference on Machine Learning (ICML-97)*, pages 170–178. Morgan Kaufmann Publishers, 1997.

[88] J. Komorowski, A Skowron, and A. Øhrn. ROSETTA rough set software system. In W Klösgen and J. Zytkow, editors, *Handbook of Data Mining and Knowledge Discovery*, pages 554–559. Oxford University Press, 2002.

[89] Robert R. Korfhage. *Information Storage and Retrieval.* Wiley, 1997.

[90] Astrid Lægreid, Torgeir R. Hvidsten, Herman Midelfart, Jan Komorowski, and Arne K. Sandvik. Predicting Gene Ontology biological process from temporal gene expression patterns. *Genome Research*, 13(5):965–979, 2003.

[91] Alex E. Lash, Carolyn M. Tolstoshev, Lukas Wagner, Gregory D. Schuler, Robert L. Strausberg, Gregory J. Riggins, and Stephen F. Altschul. SAGEmap: A public gene expression resource. *Genome Research*, 10(7):1051–1060, 2000.

[92] Seongeun Lee, Myungin Baek, Hankwang Yang, Yung-Jue Bang, Woo Ho Kim, Ji-Hong Ha, Dae-Kee Kim, and Doo-Il Jeoung. Identification of genes differentially expressed between gastric cancers and normal gastric mucosa with cDNA microarrays. *Cancer Letters*, 184:197–206, 2002.

[93] Simon M. Lin and Kimberly F. Johnson, editors. *Methods of Microarray Data Analysis.* Kluwer Academic Publishers, 2002.

[94] D. A. B. Lindberg, B. L. Humphreys, and A. T. McCray. The unified medical language system. *Methods of Information in Medicine*, 32:281–291, 1993.

[95] Robert J. Lipshutz, Stephen P.A. Fodor, Thomas R. Gingeras, and David J. Lockhart. High density synthetic oligonucleotide arrays. *Nature Genetics*, 21:20–24, January 1999. Supplement.

[96] John Wylie Lloyd. *Foundations of Logic Programming.* Springer-Verlag, second, extended edition, 1987.

[97] Stuart P. Lloyd. Least squares quantization i PCM. *IEEE Transcations on Information Theory*, IT-28(2):128–137, 1982. Orgininally published in 1957 as Technical Report, Bell Laboratories.

[98] David J. Lockhart, Helin Dong, Michael C. Byrne, Maximillian T. Follettie, Michael V. Gallo, Mark S. Chee, Michael Mittman, Chunwei Wang, Michiko Kobayashi, Heidi Horton, and Eugene L. Brown. Expression monitoring by hybridization to high-density oligonucleotide arrays. *Nature Biotechnology*, 14:1675–1680, December 1996.

[99] David J. Lockhart and Elizabeth A. Winzeler. Genomics, gene expression and DNA arrays. *Nature insight*, 405:827–836, June 2000.

[100] Harvey Lodish, Arnold Berk, S. Lawrence Zipursky, Paul Matsudaira, David Baltimore, and James Darnell. *Molecular Cell Biology*. W.H. Freeman and Company, 4th edition, 1999.

[101] Ingrid Lönnstedt and Terry Speed. Replicated microarray data. *Statistica Sinica*, 12(1):31–46, 2002.

[102] George F. Luger and William A. Stubblefield. *Artificial Intelligence: Structures and strategies for complex problem solving*. Prentice Hall, second edition, 1993.

[103] Lian-Xin Lui, Zhi-Hua Lui, Hong-Chi Jiang, Xin Qu, Wei-Hui Zhang, Lin-Feng Wu, An-Long Zhu, Xiu-Qin Wang, and Min Wu. Profiling of diffentially expressed genes in human gastric carcinoma by cDNA expression array. *World Journal of Gastroenterology*, 8(4):580–585, 2002.

[104] Myles L. Mace, Jr., Jean Mantagu, Stanley D. Rose, and Greg MacGuiness. Novel microarray printing and detection technologies. In Schena [150], pages 39–64.

[105] J. MacQueen. Some methods for classification and analysis of multivariate observations. In L.M. LeCam and J. Neyman, editors, *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297. University of California Press, 1967.

[106] Jan Małuszyński and Aida Vitoria. Towards rough datalog: Embedding rough sets in prolog. In S.K. Pal, L. Polkowski, and A. Skowron, editors, *Rough-Neuro Computing*, AI series. Springer-Verlag, 2002. To appear.

[107] Andrew Marshall and John Hodgson. DNA chips: An array of possibilities. *Nature Biotechnology*, 16(1):27–31, 1998.

[108] Eliot Marshall. Do-it-yourself gene watching. *Science*, 286(5439):444–447, 1999.

[109] Andrew K. McCallum, Ronald Rosenfeld, Tom M. Mitchell, and Andrew Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. In Jude W. Shavlik, editor, *Proceedings of the 15th International Conference on Machine Learning (ICML-98)*, pages 359–367. Morgan Kaufmann Publishers, 1998.

[110] H.W. Mewes, K. Albermann, M. Bähr, D. Frishman, A. Gleissner, J. Hani, K. Heumann, K. Kleine, A. Maierl, S.G. Oliver, F. Pfeiffer, and A. Zollner. Overview of the yeast genome. *Nature*, 387(6632):7–8, 29 May 1997. Supplement.

[111] Herman Midelfart. A bounded search space of clausal theories. In Sašo Džeroski and Peter Flach, editors, *Proceedings of the 9th International Workshop on Inductive Logic Programming (ILP-99)*, number 1634 in Lecture Notes in Artificial Intelligence, pages 210–221. Springer-Verlag, 1999.

[112] Herman Midelfart and Jan Komorowski. A rough set approach to inductive logic programming. In Wojciech Ziarko and Yiyu Yao, editors, *Proceedings of the 2th International Conference on Rough Sets and Current Trends in Computing (RSCTC-2000)*, number 2005 in Lecture Notes in Artificial Intelligence, pages 190–198. Springer-Verlag, 2000.

[113] Herman Midelfart and Jan Komorowski. A rough set approach to learning in a directed acyclic graph. In James F. Peters, Andrzej Skowron, and Ning Zhong, editors, *Proceedings of the 3th International Conference on Rough Sets and Current Trends in Computing (RSCTC-2002)*, number 2475 in Lecture Notes in Artificial Intelligence, pages 144–155. Springer-Verlag, 2002.

[114] Herman Midelfart, Jan Komorowski, Kristin G. Nørsett, Fekadu Yadetie, Arne K. Sandvik, and Astrid Lægreid. Learning rough set classifiers from gene expression and clinical data. *Fundamenta Informaticae*, 53(2):155–183, 2002.

[115] Herman Midelfart, Astrid Lægreid, and Jan Komorowski. Classification of gene expression data in an ontology. In Jose Crespo, Victor Maojo, and Fernando Martin, editors, *Proceedings of the 2nd International Symposium on Medical Data Analysis (ISMDA-2001)*, number 2199 in LNCS, pages 186–194. Springer-Verlag, 2001.

[116] Melanie Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, 1996.

[117] Tom M. Mitchell. Conditions for the equivalence of hierarchical and non-hierarchical bayesian classifiers.
http://www.cs.cmu.edu/~tom/hierproof.ps.

[118] Tom M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[119] Hung Son Nguyen and Sinh Hoa Nguyen. Discretization methods in data mining. In Polkowski and Skowron [138], pages 451–482.

[120] Sinh Hoa Nguyen and Hung Son Nguyen. Some efficient algorithms for rough set methods. In *Proceedings of the 6th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU-96)*, volume 2, pages 1451–1456, Granada, Spain, July 1-5 1996.

[121] Son Hoa Nguyen and Andrzej Skowron. Quantization of real value attributes, rough set and boolean reasoning approach. In *Proceedings of the 2nd Joint Conference on Information Sciences*, pages 34–37, Wrightsville Beach, NC, USA, October 1995.

[122] Son Hoa Nguyen and Andrzej Skowron. Quantization of real value attributes: Rough set and boolean reasoning approach. *Bulletin of International Rough Set Society*, 1(1):5–16, 1997.

[123] Ulf Nilsson and Jan Małuszyński. *Logic, Programming and Prolog.* Wiley, 1990.

[124] Kristin G. Nørsett, Astrid Lægreid, Herman Midelfart, Fekadu Yadetie, Sture Falkmer, Jon E. Grønbech, Helge L. Waldum, Jan Komorowski, and Arne K. Sandvik. Gene expression based classification of gastric carcinoma. Submitted, 2003.

[125] Hiroyuki Ogata, Susumu Goto, Kazushige Sato, Wataru Fujibuchi, and Hidemasa Bono. Kegg: Kyoto encyclopedia of genes and genomes. *Nucleic Acids Research*, 27(1):29–34, 1999.

[126] A. Øhrn, J. Komorowski, A. Skowron, and P. Synak. The design and implementation of a knowledge discovery toolkit based on rough sets: The ROSETTA system. In Polkowski and Skowron [138], pages 376–399.

[127] Aleksander Øhrn. *ROSETTA Technical Reference Manual*, March 2000. **http://www.idi.ntnu.no/~aleks/rosetta/help/manual.pdf**.

[128] Ross Overbeek, Niels Larsen, Gordon D. Pusch, Mark D'Souza, Evgeni Selkov Jr, Nikos Kyrpides, Michael Fonstein, Natalia Maltsev, and Evgeni Selkov. WIT: Integrated system for high-throughput genome sequence analysis and metabolic reconstruction. *Nucleic Acids Research*, 28(1):123–125, 2000.

[129] Packard BioScience. *QuantArray Microarray Analysis Software Manual*, 2001.

[130] Tomi Pastinen, Ants Kurg, Andrea Metspalu, Leena Peltonen, and Ann-Christine Syvänen. Minisequencing: A specific tool for DNA analysis and diagnostics on oligonucleotide arrays. *Genome Research*, 7(6):606–674, 1997.

[131] Tomi Pastinen, Mirja Raitio, Katarina Lindroos, Päivi Tainola, Leena Peltonen, and Ann-Christine Syvänen. A system for specific, high-throughput geno typing by allele-specific primer extension on microarrays. *Genome Research*, 10(7):1031–1042, 2000.

[132] Paul Pavlidis, Jason Weston, Jinsong Cai, and William Noble Grundy. Gene functional classification from heterogeneous data. In *Proceedings of the 5th International Conference on Computational Molecular Biology (RECOMB-01)*, pages 242–248, 2001.

[133] Zdzisław Pawlak. Rough sets. *International Journal of Computer and Information Sciences*, 11(5):341–356, 1982.

[134] Zdzisław Pawlak. *Rough Sets: Theoretical Aspects of Reasoning about Data.* Series D: System Theory, Knowledge Engineering and Problem Solving. Kluwer Academic Publishers, 1991.

[135] Zdzisław Pawlak. Rough set elements. In Polkowski and Skowron [138], pages 10–30.

[136] Zdzisław Pawlak and Andrzej Skowron. Rough membership functions: A tool for reasoning with uncertainty. In Cecylia Rauszer, editor, *Algebraic Methods in Logic and Computer Science.* Banach Center Publications, Volume 28, Polish Academy of Sciences, Warsaw, 1993.

[137] Charles M. Perou, Therese Sørlie, Michael B. Eisen, Matt van de Rijn, Stefanie S. Jeffery, Christian A. Rees, Jonathan R. Pollack, Douglas T. Ross, Hilde Johnsen, Lars A. Akslen, Eystein Fluge, Alexander Pergamenschikov, Cheryl Williams, Shirley X. Zhu, Per E. Lønning, Anne-Lise Børresen-Dale, Patrick O. Brown, and David Botstein. Molecular portraits of human breast tumours. *Nature*, 406(6797):747–752, 17 August 2000.

[138] Lech Polkowski and Andrzej Skowron, editors. *Rough Sets in Knowledge Discovery 1: Methodology and Applications*, volume 18 of *Studies in fuzziness and soft computing.* Physica-Verlag, Heidelberg, Germany, 1998.

[139] J.R. Pollack, T. Sørlie, C.M. Perou, C. Res, M. Eisen, S. Jeffrey, H. Johnsen, P-E. Lønning, D. Botstein, A.-L. Børresen-Dale, and P.A. Brown. Microarray analysis reveals a major direct role of DNA copy number alteration in the transcriptional program of human breast tumors. *Proceedings of the National Academy of Sciences, USA*, 99(20):12963–12968, 2002.

[140] William K. Purves, David Sadava, Gordon H. Orians, and H. Craigh Heller. *Life: The Science of Biology.* Sinauer Associates and W.H. Freeman and Company, 6th edition, 2001.

[141] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[142] Latha Ramdas, Kevin R. Coombes, Keith Baggerly, Lynne Abruzzo, W. Edward Highsmith, Tammy Krogmann, Stanley R. Hamilton, and Wei Zhang. Sources of nonlinearity in cDNA microarray expression measurements. *Genome Biology*, 2(11):research0047.1–research0047.7, 2001.

[143] Monica Riley. Functions of the gene products of *Escherichia Coli. Microbiology and molecular biology reviews*, 57(4):862–952, 1993.

[144] Stuart C. G. Rison, T. Charles Hodgman, and Jant M. Thornton. Comparison of functional annotation schemes for genomes. *Functional Integrative Genomics*, 1(1):56–69, 2000.

[145] Don Rose. Microfludic technologies and instrumentation for printing DNA microarrays. In Schena [150], pages 19–38.

[146] Miguel E. Ruiz and Padmini Srinivasan. Hierarchical neural networks for text categorization (poster abstract). In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR-99)*, pages 281–282. ACM Press, 1999.

[147] Stuart Russell and Peter Norvig. *Artificial intelligence: A modern approach.* Prentice Hall, 1995.

[148] Scanalytics, Inc, 8550 Lee Highway, Suite 400, Fairfax, VA 22031-1515, USA. *MicroArray Suite*, 2.0 edition, january 2001.

[149] M. Schena, D. Shalon, R. Davis, and P. Brown. Quantitative monitoring of gene expression patterns with a complementary DNA microarray. *Science*, 270(5235):467–470, 20 October 1995.

[150] Mark Schena, editor. *Microarray Biochip Technology.* Easton Publishing, 2000.

[151] Mark Schena, Renu A. Heller, Thomas P. Theriault, Ken Konrad, Eric Lachenmeier, and Ronald W. Davis. Microarrays: Biotechnology's discovery platform for functional genomics. *Trends in Biotechnology*, 16(7):301–306, 1998.

[152] Manfred Schmidt-Schlauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[153] Johannes Schuchhardt, Dieter Beule, Arif Malik, Eryc Wolski, Holger Eickhoff, Hans Lehrach, and Hanspeter Herzel. Normalization strategies for cDNA microarrays. *Nucleic Acids Research*, 28(10):e47, 2000.

[154] Glenn Shafer. *Mathematical Theory of Evidence.* Princeton University Press, 1976.

[155] Ron Shamir, Satoru Miyano, Sorin Istrail, Pavel Pevzner, and Michael Waterman, editors. *Proceedings of the 4th Annual International Conference on Computational Molecular Biology (RECOMB-2000).* ACM Press, 2000.

[156] Hagit Shatkay, Stephen Edwards, W. John Wilbur, and Mark Boguski. Genes, themes and microarrays: Using information retrieval for large-scale gene analysis. In *Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology (ISMB-2000)*, pages 317–328. AAAI Press, 2000.

[157] Richard Simon, Michael D. Radmacher, Kevin Dobbin, and Lisa M. McShane. Pitsfalls in the use of DNA microarray data for diagnostic and prognostic classification. *Journal of the National Cancer Institute*, 95(1):14–18, 2003.

[158] Andrzej Skowron. Boolean reasoning for decision rules generation. In J. Komorowski and Z. W. Ras, editors, *Proceedings of the 7th International Symposium on Methodologies for Intelligent Systems (ISMIS-93)*, number 689 in Lecture Notes in Artificial Intelligence, pages 295–305. Springer-Verlag, 1993.

[159] Andrzej Skowron and Jerzy W. Grzymala-Busse. From rough set theory to evidence theory. In R.R. Yager, M. Fedrizzi, and J. Kacprzyk, editors, *Advances in the Dempster-Shafer Theory of Evidence*, pages 193–236. John Wiley and Sons, 1994.

[160] Andrzej Skowron and Lech Polkowski. Synthesis of decision systems from data tables. In T.Y. Lin and N. Cercone, editors, *Rough sets and data mining: Analysis of imprecise data*, pages 259–299. Kluwer Academic Publisher, 1997.

[161] Andrzej Skowron and Cecylia Rauszer. The discernibility matrices and functions in information systems. In *Intelligent decision support: Handbook of Applications and Advances of Rough Sets Theory* [163], pages 331–362.

[162] Donna K. Slonim, Pablo Tamayo, Jill P. Mesirov, Todd R. Golub, and Eric S. Lander. Class prediction and discovery using gene expression data. In Shamir et al. [155], pages 263–272.

[163] Roman Słowiński. *Intelligent decision support: Handbook of Applications and Advances of Rough Sets Theory.* Kluwer Academic Publishers, 1992.

[164] P.H.A. Sneath. The application of computers to taxonomy. *Journal of General Microbiology*, 17:201–226, 1957.

[165] R. Sokal and C. Mitchener. A statistical method for evaluation systematic relationships. *University of Kansas Science Bulletin*, 38:1409–1438, 1958.

[166] Therese Sørlie, Charles M. Perou, Robert Tibshirani, Turid Aas, Stephanie Geisler, Hilde Johnsen, Trevor Hastie, Michael Eisen, Matt van de Rijn, Stefanie S. Jeffery, Thor Thorsen, Hanne Quist, John C. Matese, Patrick O. Brown, David Botstein, Per Eystein Lønning, and Anne-Lise Børresen-Dale. Gene expression patterns of breast carcinomas distinguish tumor subclasses with clinical implications. *Proceedings of the National Academy of Sciences, USA*, 98(19):10869–10874, 2001.

[167] Edwin Southern, Kalim Mir, and Mikhail Shchepinov. Molecular interaction on microarrays. *Nature Genetics*, 21:5–9, January 1999. Supplement.

[168] John F. Sowa. *Knowledge Representation: Logical, philosophical, and computational foundations.* Brooks/Cole, 2000.

[169] Paul T. Spellman, Gavin Sherlock, Michael Q. Zhang, Vishwanath R. Iyer, Kirk Anders, Michael B. Eisen, Patrick O. Brown, David Botstein, and Bruce Futcher. Comprehensive identification of cell cycle-regulated genes of the yeast *Saccharomyces cerevisiae* by microarray hybridization. *Molecular Biology of the Cell*, 9(12):3273–3297, 1998.

[170] V. Sperschneider and G. Antoniou. *Logic: A foundation for computer science.* Addison-Wesley, 1991.

[171] Robert Stevens, Carole A. Goble, and Sean Bechhofer. OILing the way to machine understandable bioinformatics resources. *IEEE Transactions on Information Technology and Biomedicine*, 6:129–134, 2002.

[172] Jes Stollberg, Johann Urschitz, Zsolt Urban, and Charles D. Boyd. A quantitative evaluation of SAGE. *Genome Research*, 10:1241–1248, 2000.

[173] John A. Swets. Measuring the accuracy of diagnostic systems. *Science*, 240:1285–1293, 1998.

[174] Neeme Tõnisson, Ants Kurg, Lõhmussaar, and Andres Mespalu. Arrayed primer extension on the DNA chip: Method and applications. In Schena [150], pages 247–263.

[175] Pablo Tamayo, Donna Slonim, Jill Mesirov, Qing Zhu, Sutisak Kitareewan, Ethan Dmitrovsky, Eric S. Lander, and Todd R. Golub. Interpreting patterns of gene expression with self-organizing maps: Methods and applications to hematopoietic differentiation. *Proceedings of the National Academy of Sciences, USA*, 96(6):2907–2912, 1999.

[176] Saeed Tavazoie, Jason D. Hughes, Campbell Michael J., Raymond J. Cho, and George M. Church. Systematic determination of genetic network architecture. *Nature Genetics*, 22:281–285, 1999.

[177] Robert Tibshirani, Trevor Hastie, and Michael Eisen. Clustering methods for the analysis of DNA microarray data. Technical report, Dept. of Statistics, Stanford University, October 15 1999. **http://www-stat.stanford.edu/~tibs/research.html**.

[178] George C. Tseng, Min-Kyu Oh, Lars Rohlin, James C. Liao, and Wing Hong Wong. Issues in cDNA microarray analysis: Quality filtering, channel normalization, models of variations and assessment of gene effects. *Nucleic Acids Research*, 29(12):2549–2557, 2001.

[179] Virgina Goss Tusher, Robert Tibshirani, and Gilbert Chu. Significance analysis of microarrays applied to the ionizing radiation response. *Proceedings of the National Academy of Sciences, USA*, 98(9):5116–5121, 2001.

[180] Sanjay Tyagi. Taking a census of mRNA populations with microbeads. *Nature Biotechnology*, 18(6):597–598, 2000.

[181] Vladimir Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, 1996.

[182] Victor E. Velculescu, Bert Vogelstein, and Kenneth W. Kinzler. Analysing uncharted transcriptiomes with SAGE. *Trends in Biotechnology*, 16(10):423–425, 2000.

[183] Victor E. Velculescu, Lin Zhang, Bert Vogelstein, and Kenneth W. Kinzler. Serial analysis of gene expression. *Science*, 270(5235):484–487, 20 October 1995.

[184] Jaak Vilo, Alvis Brazma, Inge Jonassen, Alan Robinson, and Esko Ukkonen. Mining for putative regulatory elements in the yeast genome using gene expression data. In Russ Altman, Timothy L. Bailey, Philip Bourne, Michael Gribskov, Thomas Lengauer, and Ilya N. Shindyalov, editors, *Proceedings for the 8th international Conference on Intelligent Systems for Molecular Biology (ISMB-2000)*, pages 384–394, Menlo Park, CA, August 16–23 2000. AAAI Press.

[185] Staal Vinterbo and Aleksander Øhrn. Minimal approximate hitting sets and rule templates. *International Journal of Approximate Reasoning*, 25(2):123–143, 2000.

[186] Ognjenka Goga Vukmirovic and Shirley M. Tilghman. Exploring genome space. *Nature insight*, 405:820–822, 2000.

[187] Michael G. Walker, Wayne Volkmuth, Einat Sprinzak, David Hodgson, and Tod Klingler. Prediction of gene function by genome-scale expression analysis: Prostate cancer-associated genes. *Genome Research*, 9(12):1198–1203, 1999.

[188] R. E. Walpole, R. H. Myers, and S. L. Myers. *Probability and statistics for engineers and scientists.* Prentice Hall, 1998.

[189] Webster's encyclopedic unabridged dictionary of the english language. Gramercy Books, 1996.

[190] A. Weigend, E. Wiener, and J. Pedersen. Exploiting hierarchy in text categorization. *Information Retrieval*, 1:193–216, 1999.

[191] Patrick Henry Winston. *Artificial Intelligence.* Addison-Wesley, third edition, 1993.

[192] Jakub Wróblewski. Finding minimal reducts using genetic algorithms. In *Proceedings of Second International Joint Conference on Information Sciences*, pages 186–189, 1995.

[193] Jakub Wróblewski. Genetic algorithms in decomposition and classification problems. In Lech Polkowski and Andrzej Skowron, editors, *Rough Sets in Knowledge Discovery 2: Applications, Case Studies and Software Systems*, volume 19 of *Studies in Fuzziness and Soft Computing*, pages 471–487. Physica-Verlag, Heidelberg, Germany, 1998.

[194] Eric P. Xing and Richard M. Karp. CLIFF: Clustering for high-dimensional microarray data via iterative feature filtering using normalized cuts. In Brunak and Krogh [22], pages S306–S315.

[195] Yee Hwa Yang, Michael J. Buckley, Sandrine Dudoit, and Terence P. Speed. Comparison of methods for image analysis on cDNA microarray data. Technical Report 584, Dept. of Statistics, University of California, Berkeley, 367 Evans Hall, Berkeley, CA 94720-3860, USA, 2000. **http://www.stat.berkeley.edu/users/terry/zarray/Html/papersindex.html**.

[196] Yee Hwa Yang, Sandrine Dudoit, Percy Luu, and Terence P. Speed. Normalization for cDNA microarray data. In Bittner et al. [13].

[197] Wojciech Ziarko. Variable precision rough set model. *Journal of Computer and System Sciences*, 46:39–59, 1993.

[198] Alexander Zien, Thomas Aigner, Ralf Zimmer, and Thomas Lengauer. Centralization: A new method for the normalization of gene expression data. In Brunak and Krogh [22], pages S323–S331.