

Filtering Large Propositional Rule Sets While Retaining Classifier Performance

Siv. ing. (MSc) Thesis

Thomas Ågotnes

Knowledge Systems Group
Department of Computer and Information Science
Norwegian University of Science and Technology

February 18, 1999

It is vain to do with more what can be done with less

William of Occam

Abstract

Data mining is the problem of inducing models from data. Models have both a descriptive and a predictive aspect. Descriptive models can be inspected and used for knowledge discovery. Models consisting of decision rules – such as those produced by methods from Pawlak’s rough set theory – are in principle descriptive, but in practice the induced models are too large to be inspected. In this thesis, extracting descriptive models from already induced complex models is considered. According to the principle of Occam’s razor, the simplest of two models both consistent with the observed data should be chosen. A descriptive model can be found by simplifying a complex model while retaining predictive performance. The approach taken in this thesis is rule filtering; post-pruning of complete rules from a model.

Two methods for finding high-performance subsets of a set of rules are investigated. The first is to use a genetic algorithm to search the space of subsets. The second method is to create an ordering of a rule set by sorting the rules according to a quality measure for individual rules. Subsets with a particular cardinality and expected good predictive performance can then be constructed by taking the first rules in the ordering. Algorithms for the two methods have been implemented and is available for general use in the ROSETTA system, a toolkit for data analysis within the framework of rough set theory. Predictive performance is estimated using ROC analysis, and ten different formulas from the literature that can be used to define rule quality are implemented. An extensive experiment on a real-world data set describing patients with suspected acute appendicitis is included. In this study, rule sets consisting of six to twelve rules with no significantly different estimated predictive performance compared to full models consisting of between 400 and 500 rules were found. Another experiment confirms these results. In the experiments, statistical hypothesis testing was used to assert difference between performance measures derived from ROC analysis.

Acknowledgments

The work in this thesis has been done in a stimulating research environment in the Knowledge Systems Group. First of all, I would like to thank my supervisor Aleksander Øhrn for his time, guidance and advice, and especially for providing the ROSETTA library. Thanks to Professor Jan Komorowski for advise, discussions and support. Thanks to Terje Løken and Tor-Kristian Jenssen for general discussions and much needed coffee breaks, and especially to Terje for – often long and heated – discussions regarding rough sets, classification and statistical hypothesis testing. Thanks to Staal Vinterbo for providing his genetic algorithm implementation, and for enthusiasm and encouragement in the early phases. Thanks to Turid Follestad for a discussion about statistical hypothesis testing, and to Keith Downing for advise regarding genetic algorithms. Thanks to Grethe Røed for commenting on a draft version of this document.

I would also like to thank those who made the experiments possible; Ulf Carlin for doing the initial analysis of the acute appendicitis data, Stein Halland et. al. for providing the acute appendicitis data, and Robert Detrano for providing the Cleveland data set.

Finally, thanks to my wife Cecilie for her support and understanding, especially during the last hectic weeks.

Contents

Acknowledgments	vii
1 Introduction	1
1.1 Knowledge Discovery in Databases	1
1.2 Model Representation	2
1.2.1 Rule-based Models	2
1.3 Motivation	2
1.3.1 Rule Pruning	3
1.3.2 Finding Submodels	3
1.4 Research Methodology and Results	4
1.5 Related Work	4
1.5.1 Pruning	4
1.5.2 Rough Set Approaches	5
1.6 Overview	5
I Background	7
2 Information and Discernibility	9
2.1 Introduction	9
2.2 Knowledge	10
2.3 Information Systems	11
2.3.1 Discretization	12
2.4 Discernibility	12
2.5 Reducts	14
2.5.1 Dynamic Reducts	14
2.6 Classifiers	15
2.6.1 Binary Classification	16
3 Propositional Decision Rules	17
3.1 Introduction	17
3.2 Syntax	17
3.3 Semantics	18
3.4 Numerical Measures	19
3.5 Generating Rules From Reducts	21
3.6 Binary Decision Rule Classifiers	21
3.6.1 Firing	22
3.6.2 Voting	22
4 Comparing Classifiers	23
4.1 Introduction	23
4.2 Testing Methodology	23

4.2.1	Desired Properties of the Test Data	24
4.3	Performance Summary	25
4.4	Accuracy	25
4.4.1	Comparing Accuracy	26
4.5	ROC Analysis	26
4.5.1	The Area Under the ROC Curve	27
4.5.2	Comparing AUC	28
5	Genetic Algorithms	31
5.1	Search as an Optimization Problem	31
5.1.1	Iterative Improvement Algorithms	33
5.2	Informal Introduction to Genetic Algorithms	33
5.3	Genetic Algorithms Defined	34
5.3.1	Concepts	34
5.3.2	A Genetic Algorithm	37
5.4	Genetic Algorithms and Optimization Problems	38
5.5	Theoretical Foundations	39
II	Rule Filtering	41
6	Rule Filtering	43
6.1	Model Pruning	43
6.2	Rule Filtering	44
6.2.1	Complexity	45
6.2.2	Performance	45
6.3	Evaluating Submodels	45
6.4	Searching	46
6.5	Comments	47
7	Quality-Based Filtering	49
7.1	Introduction	49
7.2	Defining Rule Quality	49
7.2.1	Rule Quality Formulas	50
7.2.2	Discussion	52
7.3	Quality-Based Rule Filtering	54
7.3.1	Model Selection	56
7.3.2	Undefined Quality	57
7.3.3	Resolution	57
7.4	The Search Space	58
8	Genetic Filtering	59
8.1	Introduction	59
8.2	Coding	59
8.3	The Fitness Function	60
8.3.1	General Form	60
8.3.2	Selecting a Performance Bias	60
8.3.3	Using a Cutoff-Value	62
8.3.4	Scaling	63
8.4	The Algorithm	64
8.4.1	Initialization	65
8.4.2	Resampling	66
8.4.3	Parent Selection	66
8.4.4	Genetic Operations	66

8.4.5	Recombination	66
8.4.6	Statistics	67
8.4.7	The Stopping Criterion	67
8.5	Customization	67
III	Results and Discussion	69
9	Predicting Acute Appendicitis	71
9.1	Data Material	71
9.1.1	Attributes	72
9.1.2	Data Partitioning	73
9.1.3	Surgeons' Probability Estimates	74
9.2	Initial Learning	74
9.3	Motivation	75
9.3.1	Goals	76
9.4	Rule Quality	76
9.4.1	Quality Distributions	77
9.5	Genetic Computations	77
9.5.1	Parameter Settings	77
9.5.2	GA Performance	80
9.5.3	The Search Space	82
9.6	Rule Filtering	82
9.6.1	Model Selection	83
9.6.2	Further Comparisons	93
9.7	Discussion	95
10	Predicting Coronary Artery Disease	97
10.1	Introduction	97
10.2	Data Material	97
10.2.1	Attributes	98
10.3	Preprocessing	99
10.3.1	Missing Values	99
10.3.2	Splitting and Discretizing	99
10.4	Initial Learning	100
10.5	Rule Quality	100
10.5.1	Quality Distributions	101
10.6	Genetic Computations	101
10.6.1	Parameter Settings	101
10.6.2	GA Performance	103
10.7	Rule Filtering	105
10.7.1	Model Selection	105
10.8	Discussion	113
11	Discussion and Conclusion	117
11.1	Summary	117
11.2	Discussion	117
11.2.1	Quality Based Rule Filtering	117
11.2.2	Genetic Rule Filtering	118
11.3	Conclusions	119
11.4	A Note on the Validity of the Experimental Results	120
11.5	Limitations	120
11.6	Further Work	121

A	Implementation	127
A.1	Introduction	127
A.2	Rosetta	127
A.2.1	Kernel Architecture	127
A.3	The Genetic Algorithm	128
A.3.1	Architecture	128
A.3.2	Representation	128
A.3.3	AUC Computation	129
A.3.4	User Interface	130
A.4	The Quality Based Algorithm	130
A.4.1	A Note on Rule Representation	130
A.4.2	User Interface	133
B	Quality Distributions	137
B.1	Acute Appendicitis Data	137
B.2	Cleveland Data	137

CHAPTER 1

Introduction

1.1 Knowledge Discovery in Databases

Scientists, professionals and business people have always collected empirical data. This task has been made feasible by the introduction of semiconductor electronics. The digital electronics revolution has made it possible to generate, collect, store, analyze and process increasingly large amounts of raw data. Data, information and knowledge are becoming more and more important and valuable. We are an information-based society.

Examples of domains where huge amounts of data are generated every day are the grocery trade, banking and medicine. These data are collected (generally digitized) and stored because it is expected that they contain useful and valuable patterns. The large amount of information generated by the use of bar code readers can, for instance, be used to help understand customer buying habits, or clinical data from previous patients can be used to give a medical diagnose for a current patient. Previously, data sets would often be analyzed manually, a method unfit for the huge, rapidly growing mountains of data generated today. Therefore, new methods are needed for automatic extraction of knowledge from databases by machines.

This task has been coined *Knowledge Discovery in Databases (KDD)*. One informal definition of this term has been made by Fayyad, Piatetsky-Shapiro, Smyth & Uthurusamy (1996):

Knowledge Discovery in Databases is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data.

The motivations behind, and the methods used in, KDD are common with and collected from diverse fields such as statistics, machine learning, pattern recognition and artificial intelligence. Sometimes the term *data mining* is used to describe the same task. Here, however, the view from (Fayyad et al. 1996) that data mining denotes the concrete application of algorithms for extracting patterns from data will be adopted. The goal of KDD is human *knowledge*; the goal of data mining is *patterns* in some language (generally a compact, possibly approximate, description of the data set). Formally, patterns are instantiations of some models, and data mining can be seen as searching for models and/or fitting models to data to create patterns. The KDD process uses data mining techniques to achieve it's goal. In

addition to data mining, the KDD process contains a number of steps including preprocessing of the data and interpretation and evaluation of the patterns output from the data mining step.

1.2 Model Representation

The choice of data mining technique in the KDD process affects the interpretability of the resulting patterns. Different techniques use different model representations, such as artificial neural networks, decision trees and decision rules. The representations differ both because there is a variety of data mining tasks (e.g. classification, clustering and regression) and because the goals of data mining differ. Data mining goals can be classified as being either *descriptive* or *predictive*. Descriptive models are needed for human interpretation (collecting knowledge), while the utility of predictive models is to classify new data objects automatically. The main differences between these model types are that the former demands low complexity of the models, and that the latter favors high predictive quality instead of extracting only some aspects of the data (e.g. the most important patterns).

Traditionally, data mining algorithms have targeted predictive goals. In the relatively recent appearance of the KDD field, however, descriptive models are needed to facilitate the interpretation of, and extraction of knowledge from, data. In this thesis, this problem is undertaken.

In spite of the focus on descriptive models, we emphasize that predictive models are valuable for many KDD tasks. As pointed out by Brachman & Anand (1996), the result from the data mining step, a part of a *Knowledge Discovery Support Environment (KDSE)*, can be viewed as intended to be encoded in a *KDD application*. The KDSE is used by KDD experts using domain information, while the KDD application is used by end users – typically domain experts – to extract knowledge.

1.2.1 Rule-based Models

The model representation considered in this thesis is a collection of propositional rules of the form:

$$(a_1 = v_1) \wedge (a_2 = v_2) \wedge \cdots \wedge (a_n = v_n) \rightarrow (d = v_d)$$

One alleged advantage of rule-based models is that they are of a “white-box” nature, i.e. they can readily be used for descriptive purposes. Artificial neural networks, on the other hand, have a more “black-box” nature, in that they do not provide any description of the phenomena under study other than through their input/output specification.

Rule-based models can often be used directly in KDD applications (e.g. expert systems), with little or no extra coding.

1.3 Motivation

As pointed out in the previous section, models tend to be either descriptive or predictive, and descriptive models lend themselves to direct interpretation for knowl-

edge discovery. Although the rules in a rule-based model are inherently descriptive, the model itself might not be, if it is too large. A model consisting of hundreds or thousands of rules is not very interpretable by humans and approaches a “black-box” nature. A direct cause of large sets of rules is often noisy data or rare exceptions in the collected data material. In fact, often the majority of the rules only describe a few data points rather than describing general trends.

In this thesis the problem of finding *descriptive* rule-based models is investigated. This problem includes finding the most *general* or most *important* rules, while maintaining an acceptable predictive quality. In the light of the definition of the KDD term above, the motivation is to increase the *ultimately understandable* component while retaining the *valid* component.

1.3.1 Rule Pruning

The problem of finding descriptive rule based models will be targeted by using *rule pruning*. Rule pruning is a general method for dealing with overfitting due to noise in rule based models, and can also be used to extract small models representing the most general patterns in the data.

Generally, there are two strategies to rule pruning:

- *Pre-pruning*. Pre-pruning is done during the learning phase, by using heuristic stopping criteria for determining when to stop adding more literals to a rule or more rules to a rule set.
- *Post-pruning*. Post-pruning is done after a consistent theory has been learned, and is done by simplifying the induced theory.

Pruning of propositional rules is generally done by either:

- Pruning individual rules by removing literals
- Pruning rule sets by removing whole rules

The main motivation is the principle of *Occam's razor*, stating that the simplest model that correctly fits the data should be accepted. There exist more complex than simple models, so the probability that a simple model that fits the data reflects reality is higher than the probability that a complex model that fits the data reflects reality. Or in the words of William of Occam: “it is vain to do with more what can be done by less”.

1.3.2 Finding Submodels

In this thesis rule pruning is used to extract smaller models from already induced models while maintaining an acceptable estimated prediction capability. Actually, the predictive quality (as measured on a dataset separate from the set used to generate the rules) might be expected to increase because of less overfitting. The main motivation behind this thesis is the hypothesis that there exist significantly smaller models with comparable performance.

Hypothesis 1 For a given rule-based model with high complexity, there generally exist rule-based models with significantly lower complexity and comparable predictive performance. \square

The type of rule pruning investigated in the current work, is *rule filtering*: post-pruning of whole rules. Several methods for finding subsets of a rule set with equal (or better) estimated performance will be investigated, motivated by the fact that a typical learning algorithm generates a large number of rules and that it is suspected that individual rules can be removed without lowering the performance significantly. One of the main approaches in the investigation will be trying to find a *total ordering* of the rules in a rule-based model in such a way that adding the next rule in the ordering never will decrease the estimated model performance. The existence of such orderings is the next hypothesis.

Hypothesis 2 There exists a function that orders the rules in a rule set in such a way that the estimated predictive performance of the subset consisting of the first $n + 1$ rules is larger than or equal to the estimated predictive performance of the subset consisting of the first n rules. \square

If such an ordering is found, the question is whether a model constructed by taking the first n rules in the corresponding ordering is the subset consisting of n rules with the *best* performance. In other words, even if an ordering exists, it does not mean that it is optimal.

Hypothesis 3 If hypothesis 2 is true, then the subset of size n given by the ordering has higher predictive performance than, or equal to, all other subsets with size n . \square

1.4 Research Methodology and Results

In this thesis rule filtering methods are used to filter complex models generated from real-world medical data sets. Predictive performance is assessed using ROC analysis, and statistical hypothesis testing used to compare the relative performance of two models. The main result is that it was possible to find models consisting of approximately one percent of the original rules without significantly lower performance.

1.5 Related Work

1.5.1 Pruning

A plethora of approaches to pruning sets of PROLOG like rules, propositional rule sets and decision trees is available in the machine learning literature. Esposito, Malerba & Semeraro (1993) present a common framework for post-pruning of decision trees. Quinlan (1986) uses pre-pruning in the construction of decision trees by only including leaves corresponding to a entropy based information gain over a certain threshold. Breiman, Friedman, Olshen & Stone (1984) describe methods for post-pruning decision trees, later extended by Quinlan (1987). In rule based systems, pre-pruning has been employed by Clark & Niblett (1989), Quinlan (1990) and Fürnkranz (1994) and post-pruning by Brunk & Pazzani (1991) and Cohen (1993). Fürnkranz (1997) presents algorithms for combining pre- and post-pruning in rule based systems, in addition to an overview of general pruning algorithms. Holte (1993) reports that very simple propositional rules classifying according to

one attribute only, perform well compared to other methods, and also presents results from the literature indicating that very simple models often outperform complex models. The results from Holte (1993) may be interpreted in several ways. One interpretation is that rule pruning is a good idea because good models with low complexity generally can be found, another interpretation is that rule pruning is unnecessary because univariate rules could be generated instead of complex rules in the first place. However, Holte (1993) is only concerned with predictive accuracy, and not with model descriptiveness. A model consisting of univariate rules may be small but need not be intelligible.

1.5.2 Rough Set Approaches

Here, some of the methods for finding descriptive models from rough set theory (further discussed in the next chapter) are presented. Mollestad (1997) presents a framework for extracting *default rules* reflecting the “normal” dependencies in the data. Kowalczyk (1998) presents a method for data analysis called *rough data models*, that can be used to generate small rule sets. Øhrn, Ohno-Machado & Rowland (1998b) use one particular way of ordering a rule set for rule filtering as discussed previously. Several methods for simple rule filtering is available in the ROSETTA system (Øhrn, Komorowski, Skowron & Synak 1998a). Dynamic reducts, discussed in Section 2.5.1, is a method for inducing decision rules that deals with noise in the data.

1.6 Overview

The first part of the thesis is an introduction to the theoretical frameworks used in later chapters. First, the concept of information, its representation, and how the latter can be reduced are discussed in the framework of rough set theory in Chapter 2. In Chapter 3, definitions relating to the properties of single propositional rules are presented. The above discussion illuminates the need to be able to quantitatively assert the performance of models. Asserting model performance is discussed in Chapter 4. The last chapter in the first part is an introduction to genetic algorithms, a search technique that will be used to find small models. The second part presents proposals for two algorithms for rule filtering. Chapter 6 is an introduction to rule filtering. In Chapter 7, an algorithm based on rule ordering as discussed above is presented, while a general searching method based on a genetic algorithm is presented in Chapter 8. The third and last part contains an extensive experiment analyzing real life data, found in Chapter 9. Data collected by medical doctors concerning patients with suspected acute appendicitis are used first to generate rule-based classifiers, and then pruning the induced models and comparing performance. The last part closes with a chapter with discussion and conclusions. A bibliography list is included before the appendices. Appendix A.3 contains brief presentations of the implementations of the rule filtering algorithms, and appendix B contains additional data material from the experiment in Chapter 9.

Part I

Background

Information and Discernibility

2.1 Introduction

When reasoning about knowledge discovery, a definition of the concept *knowledge* is needed. In section 2.2 one interpretation of this concept, usable for practical computational purposes, is presented. The knowledge product is used mainly for two purposes: as a model of some phenomena for human interpretation in order to gain insight about the phenomena – for example a set of rules describing the conditions when faults in some complex machinery are likely to occur – or for automatic classification of data by computers – for example in the form of an artificial neural network automatically detecting credit card fraud. In either case, the goal is to learn how to *classify* objects (for example machinery states or customers); the use of the end product may be different but the task is the same. In Section 2.6, the concept of a *classifier* is defined.

In this thesis, notions from *rough set theory* (Pawlak 1982, Pawlak 1984, Pawlak 1991) will be used to represent information and knowledge. In rough set theory, objects are characterized only by the information available about them. Objects are regarded as discernible only to the degree that the available information discerns them; if the available information about the objects is identical, the objects will be regarded as identical. A concrete notation for representation of knowledge in rough set theory is the *information system*, discussed in Section 2.3. Sections 2.4 and 2.5 briefly presents some of the machinery from rough set theory. In the next chapter, these concepts are used to generate sets of decision rules to be used as classifiers.

The current chapter is not an attempt to present a complete framework for rough set data analysis; only a brief introduction is included. The focus of this thesis is on propositional rule based systems *in general*. The methods and results are not exclusively related to rough set theory, and are applicable to other frameworks. This introduction to concepts from rough set theory is included because some of them are used for convenience in later chapters and because rough set methods are used in the experiments in Part III. The reader is referred to consult, e.g., Pawlak (1984), Pawlak & Skowron (1993) and Skowron (1993) for an introduction to these topics. See also Brown (1990) for a presentation of boolean reasoning.

2.2 Knowledge

In order to discuss how to discover knowledge, the concept itself must be defined. The term “knowledge” is the subject of debate and research in many areas, such as the cognitive sciences, philosophy, artificial intelligence and information theory. Here, we will adopt a practical view on knowledge, taken from Pawlak (1991). Pawlak argues that knowledge is closely related to *classification*. A piece of knowledge is the ability to classify, or discern between, a number of objects, situations, stimuli, etc. We will take the view that a piece of knowledge is a partition¹ of a universe $U = \{x_1, \dots, x_n\}$ – some real or abstract world represented by a finite set of *objects*. Examples of knowledge are the ability to tell blue balls from red balls (a partition of the universe of balls having two equivalence classes) and the ability to drive a car (a partition of the universe of situations consisting of the class of situations requiring turning left, the class of situations requiring braking, etc.).

The objects in the partition $U/IND(\mathbf{P})$ are here considered to be labeled in such a way that the semantics of the different classes is retained. For example, the partition $U/IND(\{Color, Fruit\})$, where U is a universe consisting of pieces of fruit, would have objects (and classes) labeled “green apples” and so on.

Example 2.1 (Knowledge Base) Consider the universe of six animals $U_l = \{x_1, x_2, x_3, x_4, x_5, x_6\}$. The available knowledge about an individual animal in U is its size, its type, its color and whether it can fly:

- Animals x_1 and x_4 are small
- Animals x_2, x_3 and x_6 are medium
- Animal x_5 is large
- Animals x_1, x_2 and x_4 are birds
- Animals x_3 and x_6 are cats
- Animal x_5 is a horse
- Animals x_1 and x_4 are white
- Animals x_2 and x_3 are black
- Animals x_5 and x_6 are brown
- Animals x_1, x_2 and x_4 are flying
- Animals x_3, x_5 and x_6 are not flying

The available knowledge can be defined as a knowledge base $K_l = (U_l, \mathbf{R}_l)$ with four equivalence relations $\mathbf{R}_l = \{Size_l, Type_l, Color_l, Flying_l\}$ over U_l , giving the following partitions:

$$U_l/Size_l = \{\{x_1, x_4\}, \{x_2, x_3, x_6\}, \{x_5\}\} \quad (2.1)$$

$$U_l/Type_l = \{\{x_1, x_2, x_4\}, \{x_3, x_6\}, \{x_5\}\} \quad (2.2)$$

$$U_l/Color_l = \{\{x_1, x_4\}, \{x_2, x_3\}, \{x_5, x_6\}\} \quad (2.3)$$

$$U_l/Flying_l = \{\{x_1, x_2, x_4\}, \{x_3, x_5, x_6\}\} \quad (2.4)$$

◇

¹Alternatively, a piece of knowledge can be defined as an equivalence relation. Since the two notions are interchangeable, both will henceforth be used.

2.3 Information Systems

Central to the rough set approach to data analysis is the concept of an *information system*. Although the mathematical notion of knowledge as partitions of a universe is theoretically appealing, a representation of knowledge for computational purposes is needed. The formal language used to represent partitions in rough set theory is data tables, tabular descriptions of the objects under consideration. An information system is a table where the rows are labeled by objects and the columns are labeled by attributes. An information system is the result of a series of observations, e.g. scientific measurements.

Definition 2.1 (Information System, Decision System). An *information system* (IS) is an ordered pair $\mathcal{A} = (U, A)$, where the *universe* U is a finite set of *objects*, and A is a finite set of *attributes* where each $a \in A$ is a total function $a : U \rightarrow V_a$. An information system $\mathcal{A} = (U, C \cup \{d\})$ is called a *decision system* if there is a distinguished attribute d which is called a *decision attribute*. Then, C is the set of *condition attributes*². \square

V_a is called the *range* of attribute a . $r(d) = |V_d|$ is called the *rank* of the decision attribute d . It will henceforth be assumed, without loss of generality³, that $V_d = \{0, \dots, r(d) - 1\}$.

A decision system is often used to represent expert classifications. If data about a number of objects (e.g. hospital patients) has been entered into an information system $\mathcal{A} = (U, A)$, this table can be extended to a decision system $\mathcal{A}' = (U, A \cup \{d\})$ where d is the decision attribute. An expert, or oracle, may assign a value to the decision attribute reflecting the experts' semantic interpretation of each object (e.g. a medical diagnosis).

The notation for an information system is illustrated in Table 2.1.

	a_1	\dots	a_m
x_1	$a_1(x_1)$	\dots	$a_m(x_1)$
\vdots	\vdots	\ddots	\vdots
x_n	$a_1(x_n)$	\dots	$a_m(x_n)$

Table 2.1: The information system $\mathcal{A} = (\{x_1, \dots, x_n\}, \{a_1, \dots, a_m\})$

Given a object $x \in U$ and an attribute $a \in A$, $a(x)$ denotes the value of the object for this attribute. Obviously, a information system represents a knowledge base. Observe that an attribute a in A corresponds to an equivalence relation over U , namely the relation

$$R = \{(x_1, x_2) \in U \times U : a(x_1) = a(x_2)\} \quad (2.5)$$

The attributes in the information system is thus a family of equivalence relations \mathbf{R} , and the decision table $\mathcal{A} = (U, A)$ represents the knowledge base $K = (U, \mathbf{R})$.

Example 2.2 (Example 2.1 continued) An information system representing the knowledge base $K_I = (U_I, \mathbf{R}_I)$ from example 2.1 is shown in Table 2.2. This

²Some authors define a decision system as a general tuple $\mathcal{A} = (U, C \cup D)$ with a set of decision attributes D . The above definition of a singleton decision attribute set is not a restriction of this definition, since any value in the set $V_{d_1} \times \dots \times V_{d_n}$ ($d_i \in D$) can be coded by the single attribute d .

³Clearly, there exists a bijection from a set V_d with rank $r(d)$ into the set $\{0, \dots, r(d) - 1\}$.

	Size	Type	Color	Flying
x_1	small	bird	white	yes
x_2	medium	bird	black	yes
x_3	medium	cat	black	no
x_4	small	bird	white	yes
x_5	large	horse	brown	no
x_6	medium	cat	brown	no

Table 2.2: The information system in Example 2.2

information system is also a decision system $\mathcal{A} = (U, \{Size, Type, Color\} \cup \{Flying\})$ with decision attribute *Flying*. \diamond

As pointed out earlier, the data in an information system is handled in a purely syntactic way, and the objects are only considered discernible up to different attribute values. The objects in a decision system can be viewed as points in an n -dimensional space (where n is the number of condition attributes) labeled with the decision d .

It is often more practical to talk about information systems and decision values than knowledge bases and partitions. In the remainder the former terms will be used, but it must be kept in mind that an information system is a description of partitions of an universe.

2.3.1 Discretization

A real-world information systems may contain attributes with continuous values. Then, all objects in the universe may be discernible, and the target equivalence relation consisting of one equivalence class per object. Thus in the rough sets framework, *discretization* of continuous attributes – combining a range of equivalence classes into one – is needed.

2.4 Discernibility

A fundamental concept in rough set theory is *discernibility*. An information system is a representation of several *indiscernibility relations*, defined over the attributes in the information system. For an information system $\mathcal{A} = (U, A)$, the indiscernibility relation $IND(B)$ corresponding to an attribute set $B \subseteq A$ is $IND(B) = \{(x, y) \in U^2 : \forall a \in B a(x) = a(y)\}$. The equivalence class corresponding to an object $x \in U$ is denoted $[x]_B$. In the following, if B is a subset of the condition attributes C from a decision system $\mathcal{A} = (U, C \cup \{d\})$, the equivalence classes in $U/IND(B) = \{E_1, \dots, E_n\}$ will be called *object classes*, and the classes in $U/IND(\{d\}) = \{X_1, \dots, X_m\}$ will be called *decision classes*. A decision system where every object class is entirely contained in a decision class is called *deterministic*. In a non-deterministic decision system, several decision values are associated with a particular object class. The set of these decision values is called the *generalized decision*, and is given by the function $\delta_C : U/IND(C) \rightarrow 2^{V_d}$. For $E \in U/IND(C)$, $\delta_C(E) = \{d(y) : y \in E\}$. The objects in an equivalence class $[x]_B \in U/IND(B)$ are indiscernible with respect to B , and *discernibility* is defined over equivalence classes rather than over

objects. Henceforth, the notation $a(E)$ will be used to refer to the value $a(x)$ for a class $E = [x]_B$. A tabulation of the attributes that discern the equivalence classes is called a *discernibility matrix*.

Definition 2.2 (Discernibility Matrix). For an information system $\mathcal{A} = (U, A)$ and attribute set $B \subseteq A$, let $U/IND(B) = \{E_1, \dots, E_n\}$. The *discernibility matrix* of \mathcal{A} is $M_{[B]} = \{m_{[B]}(i, j) : 1 \leq i, j \leq n\}$, where

$$m_{[B]}(i, j) = \{a \in B : a(E_i) \neq a(E_j)\}$$

for $1 \leq i, j \leq n$. □

Definition 2.3 (Discernibility Matrix modulo d). For a decision system $\mathcal{A} = (U, C \cup \{d\})$, let $U/IND(C) = \{E_1, \dots, E_n\}$. The *discernibility matrix modulo d* of \mathcal{A} is $M_{[C, \{d\}]} = \{m_{[C, \{d\}]}(i, j) : 1 \leq i, j \leq n\}$, where

$$m_{[C, \{d\}]}(i, j) = \begin{cases} m_{[C]}(i, j) & \text{if } \delta(E_i) \neq \delta(E_j) \\ \emptyset & \text{otherwise} \end{cases}$$

□

The discernibility matrix modulo d tabulates the condition attributes that are needed to discern between objects in the different decision classes.

To each attribute $a \in A$ we assign a unique boolean variable \tilde{a} . The set of boolean variables $\{\tilde{a} : a \in m_{[B]}(i, j)\}$ corresponding to an element in a discernibility matrix is denoted $\tilde{m}_{[B]}(i, j)$. The *discernibility functions* are formulae over these variables.

Definition 2.4 (Discernibility Functions). Given an information system $\mathcal{A} = (U, A)$ and $B \subseteq A$, let $n = |U/IND(B)|$ and let $E_k \in U/IND(B)$. The *discernibility function* of \mathcal{A} over B is

$$f_{[B]} = \bigwedge_{1 \leq i, j \leq n} \bigvee \tilde{m}_{[B]}(i, j)$$

The discernibility function of \mathcal{A} for the class E_k is

$$f_{[B]}(E_k) = \bigwedge_{1 \leq j \leq n} \bigvee \tilde{m}_{[B]}(k, j)$$

□

Definition 2.5 (Discernibility Functions modulo d). Given a decision system $\mathcal{A} = (U, C \cup \{d\})$, let $n = |U/IND(C)|$ and let $E_k \in U/IND(C)$. The *discernibility function modulo d* of \mathcal{A} is

$$f_{[C, \{d\}]} = \bigwedge_{1 \leq i, j \leq n} \bigvee \tilde{m}_{[C, \{d\}]}(i, j)$$

The discernibility function modulo d of \mathcal{A} for the object class E_k is

$$f_{[C, \{d\}]}(E_k) = \bigwedge_{1 \leq j \leq n} \bigvee \tilde{m}_{[C, \{d\}]}(k, j)$$

□

In the above definition, the notation $\bigvee S$ where S is a set of boolean variables $\{\tilde{a}_1, \dots, \tilde{a}_m\}$ is shorthand for the boolean formula $\tilde{a}_1 \vee \dots \vee \tilde{a}_m$ (similarly for $\bigwedge S$).

2.5 Reducts

The representation of the information contained in an information system can be *reduced* if not all the attributes are needed to discern between some or all of the objects. A *reduct* of a set of attributes $B \subseteq A$ from an information system $\mathcal{A} = (U, A)$ is a minimal set of attributes $B' \subseteq B$ preserving the indiscernibility relation, i.e. such that $IND(B) = IND(B')$. Reducts can also be defined for the other types of indiscernibility introduced above. The definitions of different types of reducts given below (Definition 2.6) use a concept from boolean reasoning (Brown 1990) called *prime implicants*. Any boolean function can be written in disjunctive normal form, and the set of prime implicants of the function is the set of sets consisting of the variables in each disjunct. The set of prime implicants for a boolean function f will be denoted $pri(f)$. An important result in rough set theory is that reducts are determined by prime implicants (Skowron & Rauszer 1991).

Definition 2.6 (Reducts). For the information system $\mathcal{A} = (U, A)$, the set of reducts of $B \subseteq A$ is

$$RED(B) = \{ \{a \in B : \tilde{a} \in p\} : p \in pri(f_{[B]}) \}$$

The set of reducts of B for a class $E \in U/IND(B)$ is

$$RED(E, B) = \{ \{a \in B : \tilde{a} \in p\} : p \in pri(f_{[B]}(E)) \}$$

□

Definition 2.7 (Reducts modulo d). For the decision system $\mathcal{A}' = (U, C \cup \{d\})$ with decision attribute d , the set of reducts of C modulo d is

$$RED(C, d) = \{ \{a \in B : \tilde{a} \in p\} : p \in pri(f_{[C, \{d\}]}) \}$$

The set of reducts of C for a class $E \in U/IND(C)$ modulo d is

$$RED(E, C, d) = \{ \{a \in B : \tilde{a} \in p\} : p \in pri(f_{[C, \{d\}]}(E)) \}$$

□

Reducts for a class is called *object-related* reducts. Conceptually, a reduct is the smallest set of attributes needed to define some concept. Finding reducts can be seen as *inductive learning*, i.e. learning a concept from examples. Particularly, reducts modulo the decision attribute are useful when learning *classifiers* for *decision problems*. Classifiers are briefly discussed in Section 2.6 and in the next chapter it is described how *decision rules* are generated from reducts, and how a set of decision rules can constitute a classifier. Several algorithms exist for finding reducts of information systems.

2.5.1 Dynamic Reducts

Real-world data often include noise that obscure the equivalence relations that an information system is meant to represent. One solution to dealing with noise is to find *approximations* of reducts. The approach taken by Bazan (1998) is called *dynamic reducts*. Dynamic reducts are found by computing proper reducts of a family of subtables of an information system, and selecting those reducts that occur most frequently.

Let $\mathcal{A} = (U, A)$ be an information system. A *subsystem* of \mathcal{A} is an information system $\mathcal{A}_i = (U', A)$ where $U' \subseteq U$. Let $\mathbf{P}(\mathcal{A})$ denote the set of all subsystems of \mathcal{A} . The *stability* of a set of attributes $B \subseteq A$ relative to a family of subsystems $\mathcal{F} \subseteq \mathbf{P}(\mathcal{A})$ is the fraction of the subsystems in which B is a proper reduct.

Definition 2.8 (Stability). Given an information system $\mathcal{A} = (U, A)$, the *stability* of a set of attributes $B \subseteq A$ relative to a family of subsystems $\mathcal{F} \subseteq \mathbf{P}(\mathcal{A})$ is

$$\text{stability}(B, \mathcal{F}) = \frac{|\{\mathcal{A}_i \in \mathcal{F} : B \in \text{RED}(\mathcal{A}_i)\}|}{|\mathcal{F}|}$$

If $B \in \text{RED}(\mathcal{A}_i)$ for any $\mathcal{A}_i \in \mathcal{F}$, $\text{stability}(B, \mathcal{F})$ is called the *stability coefficient* of the *generalized dynamic reduct* C relative to \mathcal{F} . \square

The stability coefficient can be used to define dynamic reducts.

Definition 2.9 ((\mathcal{F}, ε)-generalized Dynamic Reducts). Given an information system $\mathcal{A} = (U, A)$ a family of subsystems $\mathcal{F} \subseteq \mathbf{P}(\mathcal{A})$ and a number $0 \leq \varepsilon \leq 1$, the set of $(\mathcal{F}, \varepsilon)$ -generalized dynamic reducts of \mathcal{A} is

$$\text{GDR}_\varepsilon(\mathcal{A}, \mathcal{F}) = \{B \subseteq A : \text{stability}(B, \mathcal{F}) \geq 1 - \varepsilon\}$$

\square

There exist several other variations of dynamic reducts than the generalized dynamic reducts presented here. See (Bazan 1998) and (Bazan, Skowron & Synak 1994) for details.

Construction of dynamic reducts is done by iteratively sampling from a given table and computing (proper) reducts using any conventional reduct computation algorithm. For simplicity, dynamic reducts is defined above for a attribute set B only. Dynamic reducts can of course also be constructed by computing object-related reducts and/or computing reducts modulo d for a decision attribute d in a decision system, instead.

2.6 Classifiers

A *classifier* \hat{d} (Equation 2.6) over an information system $\mathcal{A} = (U, A)$ is a function that maps an object $x \in U$ to the value of the decision attribute d in a decision system $\mathcal{A}_l = (U_l, C \cup \{d\})$ used to induce the classifier.

$$\hat{d} : U \rightarrow V_d \quad (2.6)$$

Technically, there are two information systems involved in formula 2.6; the decision system \mathcal{A}_l used to create the classifier and the regular information system \mathcal{A} containing the universe that the classifier operates on. Not necessarily having a decision attribute, the objects in \mathcal{A} do not have decision *values*. It is assumed that the value set V_d in the definition of a classifier is the value set of the originating decision system. Thus, a classifier maps an object of an information system into a decision value in the classifier's originating decision system. In the context of classification, the correct (true) actual classification (decision) of an object $x \in U$ will be denoted $\bar{d}(x)$.

2.6.1 Binary Classification

The case of learning a binary classifier is of special interest. In this case, if d is the decision attribute of the originating decision system, $V_d = \{0, 1\}$. For many purposes, \hat{d} can be decomposed into two functions $\phi : U \rightarrow [0, 1]$ and $\theta : [0, 1] \rightarrow \{0, 1\}$:

$$\hat{d} = \theta \circ \phi \tag{2.7}$$

Generally, inductive learning algorithms learn the function ϕ , while the function θ is fixed for a particular algorithm. $\phi(x)$, $x \in U$, can be interpreted as the algorithm's certainty that $\bar{d}(x) = 1$ ⁴ (that the correct decision value for x is 1). Often, $\phi(x)$ estimates $\text{Prob}(\bar{d}(x) = 1)$. θ maps the certainty given by ϕ into one of the possible decision values; $\hat{d}(x) = \theta(\phi(x))$.

⁴The actual equivalence class used in defining ϕ is, of course, arbitrary, since θ can be changed accordingly if $\bar{d}(x) = 0$ is used instead of $\bar{d}(x) = 1$. We will henceforth use the convention of using the class corresponding to $\bar{d}(x) = 1$.

Propositional Decision Rules

3.1 Introduction

In the previous chapter, the problem of inductive learning was defined. The focus in this thesis is learning *propositional decision rules*. In this chapter, decision rules (Section 3.2) and their meaning (Section 3.3) will be defined, along with numerical measures of rule properties (Section 3.4). A set of decision rules is constructed by a (rule-based) inductive learning algorithm. In Section 3.5, it is described how decision rules easily can be generated from reducts. The main point of the current chapter is to show how a set of rules logically can be viewed as a binary classifier function (see Section 2.6) \hat{d} (Section 3.6).

Propositional decision rules are sentences on the form “if color equals red and size equals medium then fruit type equals apple”. Both types of goals of the knowledge discovery process – good predictive and descriptive quality – can be met by using propositional rules. One motivation when learning rules is to find rules that govern an expert’s decision making, from the examples collected in a decision system. A set of rules can be seen as a condensation of the knowledge in that decision system, each rule describing a relation between the values of some condition attributes and the expert’s decision value.

3.2 Syntax

When defining a propositional language, the atomic formulae – the propositions – first need to be defined. The atomic formulae over an information system are the set of *descriptors* (Skowron 1993); formulae on the form $(a = v)$, where a is an attribute and v is a value in V_a .

Definition 3.1 (Descriptors). Given a set of attributes B with values $V_B = \cup_{a \in B} V_a$, the set of *descriptors* $\mathcal{F}_=(B, V_B)$ over B and V_B is

$$\mathcal{F}_=(B, V_B) = \{(a = v) | a \in B, v \in V_a\}$$

The notation a_v is a syntactic shorthand for the descriptor $(a = v)$. □

Given a decision system $\mathcal{A} = (U, C \cup \{d\})$, the propositional language $\mathcal{L}_{\mathcal{A}}$ can be defined as the least set containing $\mathcal{F}_=(C \cup \{d\}, V_{C \cup \{d\}})$ and closed under the

connectives \wedge, \vee, \neg and \rightarrow . Here, we are interested only in a sublanguage of $\mathcal{L}_{\mathcal{A}}$, namely the language $\mathcal{F}_{\rightarrow}(C, d, V_C, V_d) \subset \mathcal{L}_{\mathcal{A}}$ of *decision rules* (Def. 3.3). A decision rule r is a formula on the form

$$r = \alpha \rightarrow \beta$$

where α is a conjunction of descriptors over the condition attributes and β is a single decision attribute descriptor. The next definition describes the set of conjunctive formulae.

Definition 3.2 (Conjunctive Formulae). Given a set of attributes B with values $V_B = \cup_{a \in B} V_a$, the set of *conjunctive formulae* $\mathcal{F}_{\wedge}(B, V_B)$ over B and V_B is

$$\mathcal{F}_{\wedge}(B, V_B) = \{ \bigwedge \mathcal{D} \mid \mathcal{D} \subseteq \mathcal{F}_{=}(B, V_B) \}$$

□

In the above definition, the notation $\bigwedge \mathcal{D}$ where \mathcal{D} is a set of descriptors, refers to the boolean conjunction taken over \mathcal{D} ; for example, $\bigwedge \{a_1, b_2, c_3\} = a_1 \wedge b_2 \wedge c_3$.

Note that the set of conjunctive formulae contains the set of descriptors; $\mathcal{F}_{=}(B, V_B) \subset \mathcal{F}_{\wedge}(B, V_B)$.

Definition 3.3 (Decision Rules). Given a decision system $\mathcal{A} = (U, C \cup \{d\})$, the set of *decision rules* $\mathcal{F}_{\rightarrow}(C, d, V_C, V_d)$ over \mathcal{A} is

$$\mathcal{F}_{\rightarrow}(C, d, V_C, V_d) = \{ \alpha \rightarrow \beta \mid \alpha \in \mathcal{F}_{\wedge}(C, V_C), \beta \in \mathcal{F}_{=}(V_d, V_{\{d\}}) \}$$

□

For a decision rule $r = \alpha \rightarrow \beta$, α is called the rule's *antecedent* and β is called its *consequent*.

3.3 Semantics

Decision rules are well-defined strings of symbols defined over an alphabet, and have no inherent meaning. Particularly, they are not related to their originating decision system in any other way than that the descriptors are built from the names, and value sets, of the attributes in that decision system. A descriptor ($a = v$) is not a mathematical expression involving the attribute a ; it is a string of symbols.

The meaning of decision rules and conjunctive formulae is defined relative to an information system, most often one different from the rules' originating decision system. The semantics of a descriptor ($a = v$) is the subset of the universe that has the value v for the attribute a . The semantics is only defined for information systems containing an attribute a with a value set including v , the semantics relative to other information systems is not interesting and is said to be undefined. The semantics of conjunctions with two or more conjuncts and of decision rules are defined inductively with the usual semantics of conjunction and implication.

Definition 3.4 (Semantics of Conjunctive Formulae). Given an information system $\mathcal{A} = (U, A)$, with $a \in A$ and $v \in V_a$, the semantics $\llbracket \varphi \rrbracket_{\mathcal{A}}$ of a conjunctive formula $\varphi \in \mathcal{F}_{\wedge}(A, V_A)$ in \mathcal{A} is:

$$\llbracket \varphi \rrbracket_{\mathcal{A}} = \begin{cases} \{x \in U \mid a(x) = v\} & \text{if } \varphi \equiv (a = v) \in \mathcal{F}_{=}(A, V_A) \\ \llbracket \varphi' \rrbracket_{\mathcal{A}} \cap \llbracket \varphi'' \rrbracket_{\mathcal{A}} & \text{if } \varphi \equiv \varphi' \wedge \varphi'' \end{cases}$$

□

The subscript \mathcal{A} will be omitted when it is understood from the context. Since conjunction is associative under the semantics function, that is, $\llbracket \varphi' \rrbracket \cap \llbracket \varphi'' \wedge \varphi''' \rrbracket \equiv \llbracket \varphi' \wedge \varphi'' \rrbracket \cap \llbracket \varphi''' \rrbracket$ for all $\varphi', \varphi'', \varphi''' \in \mathcal{F}_\wedge(A, V_A)$, the semantics of the conjunctive formula $\varphi' \wedge \varphi'' \wedge \varphi'''$ is well-defined.

The semantics of a decision rule is the set of objects that always satisfy the consequent when they satisfy the antecedent.

Definition 3.5 (Semantics of Decision Rules). Given a decision system $\mathcal{A} = (U, C \cup \{d\})$, the semantics $\llbracket r \rrbracket_{\mathcal{A}}$ of a decision rule $r = \alpha \rightarrow \beta \in \mathcal{F}_{\rightarrow}(C, d, V_C, V_d)$ in \mathcal{A} is defined as:

$$\llbracket r \rrbracket_{\mathcal{A}} = (U - \llbracket \alpha \rrbracket_{\mathcal{A}}) \cup \llbracket \beta \rrbracket_{\mathcal{A}}$$

□

3.4 Numerical Measures

Properties of the semantics of a decision rule relatively to an information system can be analyzed quantitatively. The most frequently used numerical measures with respect to propositional rules can be derived from the *contingency table* (Bishop, Fienberg & Holland 1991) – a 2×2 matrix. A decision rule $r = \alpha \rightarrow \beta$ gives the two binary partitions of the universe U corresponding to its originating decision system:

$$U = U_\alpha \cup U_{\neg\alpha} = \llbracket \alpha \rrbracket \cup (U - \llbracket \alpha \rrbracket) \quad (3.1)$$

$$U = U_\beta \cup U_{\neg\beta} = \llbracket \beta \rrbracket \cup (U - \llbracket \beta \rrbracket) \quad (3.2)$$

The contingency table gives the cardinalities of these four equivalence classes. Table 3.1 shows the contingency table for the decision rule $r = \alpha \rightarrow \beta$. The frequen-

	U_β	$U_{\neg\beta}$	
U_α	$n_{\alpha,\beta}$	$n_{\alpha,\neg\beta}$	n_α
$U_{\neg\alpha}$	$n_{\neg\alpha,\beta}$	$n_{\neg\alpha,\neg\beta}$	$n_{\neg\alpha}$
	n_β	$n_{\neg\beta}$	$ U $

Table 3.1: Contingency table for the decision rule $r : \alpha \rightarrow \beta$

cies in the contingency table are:

$$n_{\varphi,\psi} = |U_\varphi \cap U_\psi| \quad (3.3)$$

for $\varphi \in \{\alpha, \neg\alpha\}$ and $\psi \in \{\beta, \neg\beta\}$. The interpretation of Table 3.1 is:

- $n_{\alpha,\beta}$ denotes the number of objects that satisfy both α and β , $n_{\alpha,\neg\beta}$ the number of objects that satisfy α but not β , etc.
- $n_\alpha = n_{\alpha,\beta} + n_{\alpha,\neg\beta}$ denotes the number of objects that satisfy α , $n_{\neg\alpha} = n_{\neg\alpha,\beta} + n_{\neg\alpha,\neg\beta}$ the number of objects that do not satisfy α .
- $n_\beta = n_{\alpha,\beta} + n_{\neg\alpha,\beta}$ denotes the number of objects that satisfy β , $n_{\neg\beta} = n_{\alpha,\neg\beta} + n_{\neg\alpha,\neg\beta}$ the number of objects that do not satisfy β .
- $|U| = n_\alpha + n_{\neg\alpha} = n_\beta + n_{\neg\beta}$ is the cardinality of the universe.

Sometimes, it is useful to refer to the *relative* frequencies. In the contingency table in Table 3.2,

$$f_{\varphi,\psi} = \frac{n_{\varphi,\psi}}{|U|} \quad (3.4)$$

$$f_{\varphi} = \frac{n_{\varphi}}{|U|} \quad (3.5)$$

$$f_{\psi} = \frac{n_{\psi}}{|U|} \quad (3.6)$$

for $\varphi \in \{\alpha, \neg\alpha\}$ and $\psi \in \{\beta, \neg\beta\}$.

	U_{β}	$U_{\neg\beta}$	
U_{α}	$f_{\alpha,\beta}$	$f_{\alpha,\neg\beta}$	f_{α}
$U_{\neg\alpha}$	$f_{\neg\alpha,\beta}$	$f_{\neg\alpha,\neg\beta}$	$f_{\neg\alpha}$
	f_{β}	$f_{\neg\beta}$	1

Table 3.2: Frequency based contingency table for the decision rule $r = \alpha \rightarrow \beta$

The number of objects satisfying a conjunctive formula or a decision rule is called the formula's support.

Definition 3.6 (Support of Conjunctive Formulae and Decision Rules). The *support* of a conjunctive formula or decision rule φ in an information system \mathcal{A} is:

$$\text{support}(\varphi) = |\llbracket \varphi \rrbracket_{\mathcal{A}}|$$

□

Several properties of a decision rule relative to an information system can be calculated based on the contingency table. The *accuracy*, also called the *consistency* (Michalski 1983), of a decision rule is a measure of how correct the rule is.

Definition 3.7 (Accuracy). The *accuracy* of a decision rule r is:

$$\text{accuracy}(r) = \frac{\text{support}(\alpha \wedge \beta)}{\text{support}(\alpha)} = \frac{n_{\alpha,\beta}}{n_{\alpha}}$$

□

A decision rule has high accuracy if a high proportion of the objects satisfied by the antecedent also are satisfied by the consequent. Accuracy is an estimate, based on the frequencies given by the originating information system, of the probability $\text{Prob}(\beta|\alpha)$. It can be interpreted as the probability of making a correct classification for an object that satisfies α , with the rule.

A decision rule may only describe part of a phenomena indicated by the consequent. That is, a single rule may be able to classify only some of the objects in the equivalence class indicated by its consequent. The *coverage*, also called the *completeness* (Michalski 1983), of a decision rule is an estimate of the fraction of all the objects in the information system belonging to the indicated equivalence class, that the rule can classify into this class.

Definition 3.8 (Coverage). The *coverage* $coverage(r)$ of a decision rule r is:

$$coverage(r) = \frac{support(\alpha \wedge \beta)}{support(\beta)} = \frac{n_{\alpha, \beta}}{n_{\beta}}$$

□

A decision rule has high coverage if many of the objects satisfied by the consequent also are satisfied by the antecedent. Similar to accuracy, coverage is a frequency-based estimate of the probability $Prob(\alpha|\beta)$. It can be interpreted as the probability that an object satisfied by the consequent can be classified with the given rule.

3.5 Generating Rules From Reducts

By overlaying object-related reducts (Section 2.5) of the attributes in a decision system over the objects in the universe, minimal (in the case of proper reducts) or approximative (in the case of dynamic reducts) decision rules can easily be generated. Each set of object-related reducts attempt to discern one object class from all other classes, and the set of rules generated from all sets of object-related reducts can be used as a classifier as defined in the next section.

Let $\mathcal{A} = (U, C \cup \{d\})$ be a decision system with decision attribute d . The set $RUL(C, x, d)$ of minimal decision rules for an object $x \in U$ is

$$RUL(C, x, d) = \left\{ \bigwedge_{a \in B} (a = a(x)) \rightarrow (d = d(x)) : B \in RED([x]_C, C, d) \right\} \quad (3.7)$$

The set $RUL(\mathcal{A})$ of minimal decision rules over \mathcal{A} is

$$RUL(\mathcal{A}) = \bigcup_{x \in U} RUL(C, x, d) \quad (3.8)$$

Decision rules for dynamic reducts are generated similarly by replacing $RED([x]_C, C, d)$ in Equation 3.7 with $GDR_{\varepsilon}(\mathcal{A}, \mathcal{F})$ for some family of subsystems \mathcal{F} and a number ε .

3.6 Binary Decision Rule Classifiers

In Section 2.6 of the previous chapter, a classifier was defined as a function

$$\hat{d} : U \rightarrow V_d$$

Several strategies exist for using a set RUL of rules as a classifier, one of the most important decisions being whether to use *ordered* or *unordered* sets of rules. Using ordered rule sets is fairly straightforward; the first rule that fires (i.e. that has a matching antecedent) is used. Using unordered rule sets involves more degrees of freedom; how should several firing rules be combined to make a classification? Should some rules be considered more important than others? An example of an algorithm implementing both approaches is the CN2 algorithm (Clark & Niblett 1989).

Here, one particular strategy for classifying with unordered rule sets taken from Øhrn (1998) is presented. This scheme is used in later chapters.

Binary classifiers are viewed as a composition $\hat{d} = \theta \circ \phi$ (Section 2.6):

$$\begin{aligned}\phi : U &\rightarrow [0, 1] \\ \theta : [0, 1] &\rightarrow \{0, 1\}\end{aligned}$$

We define the function θ as a threshold function $\theta = \theta_\tau$:

$$\theta_\tau(r) = \begin{cases} 1 & \text{if } r \geq \tau \\ 0 & \text{otherwise} \end{cases} \quad (3.9)$$

To implement a classifier, the value $\phi(x)$ – the certainty that the correct classification $\hat{d}(x) = 1$ must be determined for all $x \in U$.

3.6.1 Firing

A decision rule r is said to be *firing* for an object x if the rule's antecedent matches the object. The first step in determining $\phi(x)$ is to find all the rules $RUL' \subseteq RUL$ in a rule set RUL that fires for x :

$$RUL' = \{\alpha \rightarrow \beta \in RUL \mid x \in \llbracket \alpha \rrbracket\} \quad (3.10)$$

In the case that none of the rules in RUL fires ($RUL' = \emptyset$), the value $\phi(x)$ is set to a predefined fallback certainty. Otherwise, a voting mechanism is employed.

3.6.2 Voting

Having found a non-empty set of firing rules RUL' , the strategy is to let each of the firing rules $r = \alpha \rightarrow \beta$ cast a number $votes(r)$ of *votes* in favor of the decision that $\hat{d}(x) = 1$:

$$votes(r) = support(\alpha \wedge \beta) \quad (3.11)$$

where the support is calculated over the rule set's originating information system. This scheme considers the importance of a rule to be proportional to the number of objects it matches, giving general patterns more influence than random noise.

After $votes(r)$ has been determined for all firing rules, $\phi(x)$ is calculated as follows:

$$\phi(x) = \frac{\sum_{r=\alpha \rightarrow (d=1) \in RUL'} votes(r)}{\sum_{r \in RUL'} votes(r)} \quad (3.12)$$

The accumulated number of votes in favor of $\hat{d}(x) = 1$ is divided by a normalization factor equal to the sum of all casted votes. $\phi(x)$ actually gives the percentage of all casted votes in favor of $\hat{d}(x) = 1$.

Equation 3.12 is an instantiation of a basic scheme for using a rule set as a classifier, representing the particular strategy that will be used in later chapters. Øhrn (1998) presents several modifications of this scheme; e.g. modification of the firing criterion to deal with missing values and rule hierarchies, using an equal number of votes for all rules, and normalizing over the sum of the votes for *all* rules instead of the firing rules.

Comparing Classifiers

4.1 Introduction

For validation of the hypotheses stated in the introductory chapter (page 3), a method for comparing the relative performance of rule-based classifiers is needed. The focus will henceforth be on binary classifiers. In this chapter two comparison methods are presented; the traditional *accuracy* measure (Section 4.4) and *ROC analysis* (Section 4.5). Salzberg (1997) argues that comparative studies of classifier performance must be done very carefully to avoid statistical invalidation of the experiment. Particularly, all the available data should not be looked at before, or during, parameter tuning of the algorithms. Experiment design is discussed in the next section.

4.2 Testing Methodology

For all practical purposes, an exact measure of predictive performance is unattainable. After all, if all objects in the universe can be perceived then the classifier is no longer needed because a perfect classifier that classifies every object correctly can be constructed. In practice, only a limited number of objects collected from a possibly infinite set is available. This limited dataset must be used for inductive learning, and is also the only data available to be used to obtain information about the predictive performance of the induced models. The measures discussed in this chapter will thus be *estimates* of a classifier's performance.

Experiments comparing classifiers are generally performed in order to compare different inductive learning algorithms or different parameters for the same algorithm. Since only a limited number of objects are available, the data set must be split into at least two disjunct sets. The reason for this is that a measure of predictive performance must estimate as close as possible the performance of the classifier in operation on *unseen objects*. If some of the objects used to estimate performance also are used in the inductive learning step, the classifier will become biased and the estimate will not be valid. A correct classification is needed by both the learning algorithm and the performance estimation procedure; the available data is a decision system $\mathcal{A} = (U, C \cup \{d\})$. \mathcal{A} is split into two disjunct subsets; \mathcal{A}_T and \mathcal{A}_V . \mathcal{A}_T – called the *training set* – is used for *training* which includes induc-

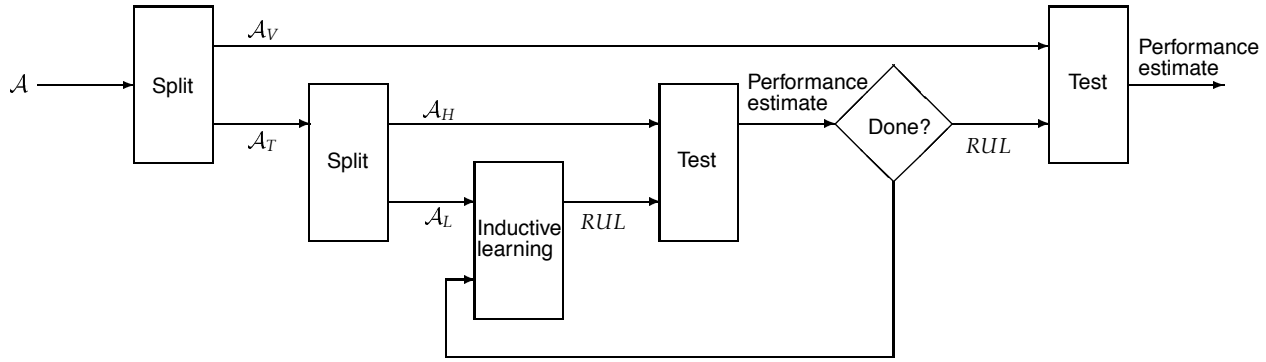


Figure 4.1: Overview of the testing methodology. Combined control/data flow.

tive learning, while \mathcal{A}_V – called the testing set – is used to estimate the performance of classifiers generated from \mathcal{A}_T . The training set is further divided into the two disjunct sets \mathcal{A}_L – henceforth called the *learning set* – and \mathcal{A}_H – called the *hold-out set*. The learning set is used as input knowledge to the inductive learning algorithm. The hold-out set is used by a general data mining algorithm to tune the parameters for the learning algorithm or to postprocess the result from the learning algorithm. Generally, the hold-out set is used to estimate the performance of classifiers resulting from various parameter settings; this can not be done with the testing set because parameter tuning is a part of the learning phase and the test data must not be used at that stage. Also, this intermediate performance estimation should not be done using the learning set to avoid overfitting¹. The splitting of the original data set into the three subsets is generally done randomly. Figure 4.1 shows an overview of the testing methodology.

In a practical experiment, some or all of the steps in Figure 4.1 are repeated for several different splits of the original data set. A systematic approach to this procedure is *cross-validation*.

The machine learning community and, more recently, the data mining community often use a set of benchmark data sets to compare the performance of various algorithms (see e.g. Holte (1993)). Many of these data sets are collected at the UC Irvine repository of machine learning databases (Murphy & Aha 1995).

4.2.1 Desired Properties of the Test Data

The validity of a performance estimate is highly dependent upon the particular data chosen for the testing set. Swets (1988) suggests four properties of an ideal testing set:

1. The decision value for each object must be correct. This is not a trivial property; in many domains it is difficult to obtain the true decision with 100% certainty.

¹ As commented earlier, a theory describing the already seen data perfectly (or near-perfectly in the case of inconsistent data) can be constructed. By setting the parameters according to the performance on data not seen in the learning step, the learning algorithm is tuned away from an extremely close fit, if necessary.

2. The determination of the correct decision values must be independent of the system under consideration; that is, the learning system may not be involved in determining the correct decision.
3. The selection of test data must not be dependent upon the particular procedures used to establish the correct decision value. For example, objects should not be included in the testing set only because they were easy to measure.
4. The testing set should reflect the universe of objects that the system is to operate on in the future. The distribution of decision values in the testing set should approximate the real distribution.

4.3 Performance Summary

The performance of a binary classifier, relative to a given decision system \mathcal{A} can be summarized in a 2×2 contingency table, called a *confusion matrix*. A confusion matrix contains information about a classifier's performance similar to the information about the performance of a propositional decision rule contained in a contingency matrix as discussed in Section 3.4. A confusion matrix for the decision system $\mathcal{A} = (U, C \cup \{d\})$ is shown in Table 4.1. \hat{d} is the classifier learned from another decision system \mathcal{A}_L . Recall from Chapter 2 that a classifier maps objects from the target information system into decision values from the originating decision system. Here, the target IS is also a decision system with decision attribute d . The entries in the confusion matrix are the numbers of objects in the target de-

		\hat{d}	
		0	1
d	0	TN	FP
	1	FN	TP

Table 4.1: A confusion matrix

cision system corresponding to the four possible combinations of values from the two binary functions. The first row in the matrix gives the number of objects with decision 0. These are divided into the number of *true negatives* (TN), the number of objects correctly classified as having decision 0, and the number of *false positives*, the number of objects falsely predicted as having decision 1. Similarly, the second row gives the numbers of *false negatives* (FN) and *true positives* (TP).

4.4 Accuracy

The, by far, most used measure for comparing predictive performance in the machine learning literature is classifier *accuracy*². The accuracy of a classifier, relative to a decision system, is defined as proportion of the objects that are correctly classified:

$$accuracy = \frac{TN + TP}{N} \quad (4.1)$$

²Or the *error rate*; one minus the accuracy.

where N is the total number of objects ($N = TN + FP + FN + TP$).

Classifier accuracy is an intuitive measure of performance. The wide use of accuracy for making conclusions regarding the relative performance of inductive learning algorithms has, however, been criticized. Observe that, in Equation 4.1, a decrease in the number of true positives can be compensated by an increase in the number of true negatives. In other words, the accuracy measure assumes equal cost for misclassifying “positive” and “negative” objects. In practice, this is often not the case. For example, the “cost” of not performing surgery on a patient with acute appendicitis is considered to be much higher than performing surgery on a patient wrongfully diagnosed as having acute appendicitis. In addition, accuracy maximization assumes that the class distribution (the a priori probabilities of the decision values) for the real population is known. Provost, Fawcett & Kohavi (1998) suggest justifications for using accuracy, and argues that “these justifications are questionable at best”.

4.4.1 Comparing Accuracy

As argued by Salzberg (1997), statistical testing of the difference between accuracy measures for classifiers must be done very carefully, and statistical invalid conclusions are common in the literature. Several authors suggest that *McNemar’s test* is the appropriate statistical test (Ripley 1996).

4.5 ROC Analysis

*Relative Operating Characteristic*³ (ROC) analysis originated in signal theory as a method for signal discrimination, and is gaining increasingly attention in the machine learning field for analyzing the discriminatory capabilities of binary classifiers. ROC analysis gives a precise and valid representation of a binary classifier’s capability of discriminating “signal” from “noise”. Here, a “signal” will be defined as an object x with correct classification $\tilde{d}(x) = 1$ ⁴.

Several authors, e.g. Swets (1988) and Provost et al. (1998), critically analyze the use of accuracy, and argue that the ROC curve is the appropriate measure. Above, accuracy was defined for a binary classifier (equation 2.6), that is, for a fixed τ in equation 3.9 for binary decision rule classifiers. ROC analysis, on the other hand, considers the performance of a range of classifiers, parameterized over the threshold value τ . Each value of τ between 0 and 1 gives rise to a confusion matrix for the corresponding classifier. A proper performance measure should, unlike the accuracy measure, not depend upon the raw frequencies in the confusion matrix. In ROC analysis, relative proportions are used instead of raw frequencies. Note that all the information in the confusion matrix is contained in only two proportions; one from each row of the matrix, each being the proportion of one of the two elements in the row relative to the sum of that row. The two proportions used in ROC analysis are the *true-positive proportion* $\frac{TP}{TP+FN}$ and the *false-positive proportion* $\frac{FP}{TN+FP}$. The former denotes the fraction of “hits”, the latter the fraction of “false alarms”. By only using these proportions for a range of thresholds, the two problems indicated above relating to the accuracy measure are solved. First, these proportions does

³Or *Receiver Operating Characteristic*, in the field of signal detection.

⁴Of course, ROC analysis can be performed relative to any decision class. See Section 2.3 for notes on the enumeration of decision values. Also, ROC analysis can be performed for non-binary classifiers by assigning a new decision value (“noise”) to the objects with decision value different from 1 (“signal”).

not depend upon the prior probabilities. For example, the true-positive proportion is comparable for data sets with different proportions of positive events ($\frac{TP+FN}{N}$). Second, the analysis does not depend upon a particular threshold τ . This allows classifiers to be compared independent upon the relative costs of the two types of error (false positives and false negatives). In addition, the threshold value selected is in practice dependent upon the prior probabilities. In ROC analysis, an *ROC curve* is plotted in a coordinate system with the values of the two proportions running along each axis. One point on the curve corresponds to one τ value. The true-positive proportion and one minus the false-negative proportion are given special names:

$$\text{Sensitivity} = \frac{TP}{TP + FN} \quad (4.2)$$

$$\text{Specificity} = \frac{TN}{TN + FP} \quad (4.3)$$

Sensitivity and specificity are thus properties of a classifier, and an ROC curve represents these measures for a range of classifiers only differing in the threshold value τ . Sample ROC curves are shown in Figure 4.2. A straight line from (0, 0) to (1, 1) indicates only random classificatory ability.

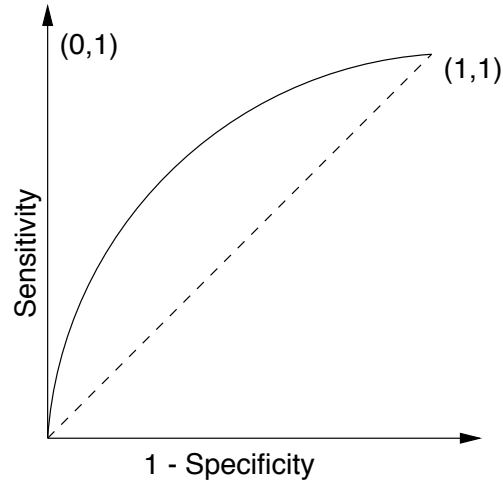


Figure 4.2: Sample ROC curves. Each point on the curves corresponds to a decision threshold. The dashed curve corresponds to a classifier with no discriminatory capability.

Although classifier performance should not be compared for a fixed τ , ROC curves can also be used to select a particular threshold value for a given application. Generally, the point closest to (0, 1) is a good choice if the two types of error have the same associated cost.

4.5.1 The Area Under the ROC Curve

Comparing accuracy measures for different classifiers measured on the same data is relatively easily, since the accuracy values are real numbers between 0 and 1. Comparing ROC curves, however, is more difficult. If the graph for one ROC curve is greater than the graph for another (the first curve lies entirely above the second),

then the first ROC curve is said to *dominate* the second. A classifier with a dominating ROC curve are always said to be better. Generally, however, ROC curves are not dominating or dominated, but intersects at one or more points. To compare the estimated performance of classifiers with general ROC curves measured on the same data, a single-valued measure representing some of the information in the ROC curve is needed. The area under the ROC curve (AUC) is generally accepted as the preferred single number measure. In the following, the AUC for a ROC curve computed for a rule-based classifier RUL relative to a decision system \mathcal{A} will be denoted

$$AUC_{\mathcal{A}}(RUL) \quad (4.4)$$

A ROC graph consists of a finite number of points, and the AUC is generally computed using the trapezoidal rule of integration. Hanley & McNeil (1982) provide expressions for approximating the standard error $se(AUC)$ corresponding to the computation of an AUC estimate AUC using the trapezoid rule.

4.5.2 Comparing AUC

Given two classifiers, ROC analysis of the classifications of a data set (a testing set or a hold-out set) can be performed in order to compute the respective measures of performance, the AUCs, for the classifiers. Collapsing the ROC curves into single numbers allows the performance of the two classifiers to be compared. But can one be certain that one classifier is better than another if the AUC is higher? Clearly, small differences in the computed AUCs can be random. To assess whether the differences between the AUCs computed from the same data set is random or real, statistical hypothesis testing can be employed.

Hanley & McNeil (1983) defines the following test statistic:

$$Z = \frac{AUC_1 - AUC_2}{se(AUC_1 - AUC_2)} \quad (4.5)$$

where AUC_1 and AUC_2 are the AUC measures for the two classifiers. It is inappropriate to compute the standard error of the difference between two AUC values computed from the same data directly. When computed from the same data, AUC_1 and AUC_2 are very likely to be correlated. In the following formula for calculating the standard error for the difference, Hanley & McNeil (1983) takes into account the correlation between the classifications made by the two classifiers:

$$se(AUC_1 - AUC_2) = \sqrt{se_1^2 + se_2^2 - 2rse_1se_2} \quad (4.6)$$

where se_1 and se_2 are the standard errors for AUC_1 and AUC_2 , respectively, and r is a value that represents the correlation between the two areas. r is a function of the average value of two intermediate correlation coefficients and of the average areas. The intermediate coefficients are the correlations between the two classifiers' certainty values for objects with negative decision and positive decision, respectively. These coefficients can be computed using either the Pearson product-moment correlation method or the Kendall tau. For a tabulation of r , we refer to Hanley & McNeil (1983).

Z (Equation 4.5) is standard normally distributed under the hypothesis that the two areas really are equal, and can be used to test – under a certain significance level – whether the two areas are statistically likely to be different. For convenience we will in the following discuss the one-tailed case, where the problem is

to conclude whether one particular classifier is better than the other or not (rather than that *one* of the classifiers is better than the other as in the two-tailed case). The following discussion is easily applied to the two-tailed case.

Consider that AUC values has been calculated for two classifiers on the same data, and that the value of the first classifier, AUC_1 , is slightly larger than the value for the second classifier, AUC_2 . The one-tailed case is used to determine whether the difference is random or real. The hypotheses for a one-tailed test are:

$$H_0 : \text{The areas are really equal, and } Z \text{ is normally distributed} \quad (4.7)$$

$$H_1 : \text{The area for the first classifier is larger than the area for the second} \quad (4.8)$$

$$\text{classifier, and } Z \text{ is not normally distributed.} \quad (4.9)$$

The hypothesis testing is done by selecting a significance level α and calculating a value, called the *critical ratio*, z of Z . The probability of a large absolute value z under the hypothesis that Z is standard normally distributed is low, and the null hypothesis H_0 should be rejected in favor of the alternative hypothesis H_1 in that case. More specifically, the null hypothesis should be rejected in the case that the p-value corresponding to the value z is lower than α or, equivalently, that the value z is higher than the Z-value z_α corresponding to the probability α . These concepts are illustrated in Figure 4.3.

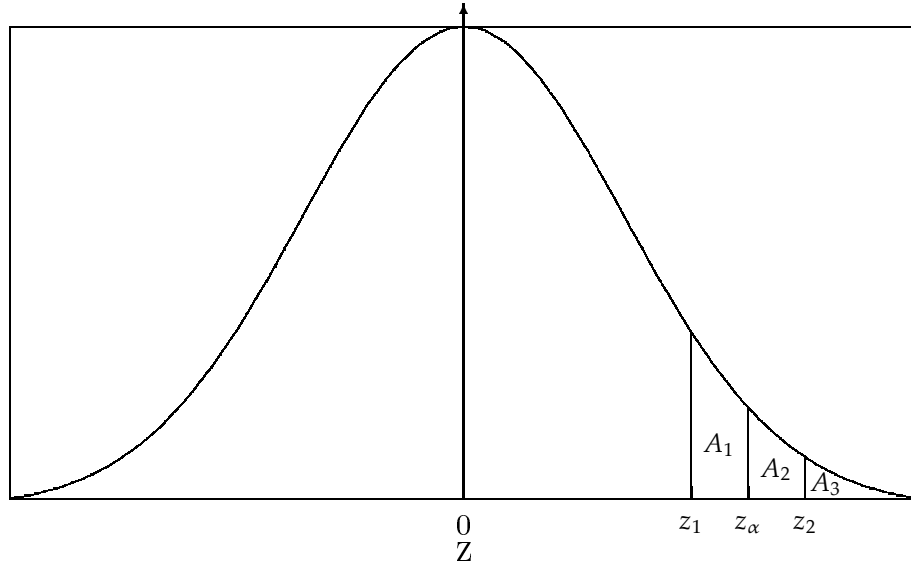


Figure 4.3: Two-tailed hypothesis testing. $p_{z_2} = A_3$, $\alpha = A_2 + A_3$ and $p_{z_1} = A_1 + A_2 + A_3$. z_1 and z_2 are calculated values of the test statistic Z . $p_{z_1} > \alpha$ ($z_1 < z_\alpha$) and the null hypothesis is not rejected, leading to the conclusion that the two areas used to calculate z_1 are not significantly different. $p_{z_2} < \alpha$ ($z_2 > z_\alpha$) and the null hypothesis is rejected, leading to the conclusion that the two areas used to calculate z_2 are statistically significant different.

The significance level α is the probability that we conclude that the difference is statistically significant, when it really is not (the probability that H_0 is rejected when it reflects the true state of nature); see Figure 4.3:

$$\alpha = \text{Prob}(Z \geq z_\alpha | H_0) \quad (4.10)$$

The lower the value of α , the more certain we can be that the difference really is significant when we conclude that it is.

Testing for Statistically Insignificant Difference

The hypothesis test discussed above is designed to determine, under a certain significance level, whether one classifier can be considered to perform – statistically speaking – better than another. The validity of the claim of significant difference depends on the value α ; the smaller α the more we can rely on the claim. Sometimes, however, it is the opposite conclusion that is interesting – namely whether one classifier can be considered to perform – statistically speaking – *not* any worse than another, that is, to have *equal* or better performance. Ideally, this problem could be attacked by exchanging the hypotheses H_0 and H_1 above and to use a test statistic with known distribution under H_1 (that the significance is different). Then, one could make the claim that the two classifiers are (at least) equally good (statistically speaking) accompanied by a certain significance level. Alas, there is no such test statistic available. Lacking a proper statistic for this “direction” of the test, Equation 4.5 will be used for this purpose in later experiments. It is very important to point out what the test described above enables us to conclude regarding the “equality” between classifiers. The case of *not* rejecting H_0 tells only that not enough evidence to support H_1 was available; it says nothing statistically about the validity of H_0 . In other words, saying that H_0 is valid on the grounds that it is not rejected is a statistically invalid conclusion. It is natural, however, to *assume* that the classifiers are equal because there is not enough evidence available to conclude that they are significantly different. The difference between drawing a statistical conclusion and assuming that the null hypothesis holds because the alternative hypothesis can not be accepted is crucial. Note that if the alternative hypothesis had specified an alternative distribution for Z (instead of “not normally”), the type II error – the probability of accepting the null hypothesis given that the alternative hypothesis is true – could be calculated to accompany the assumption of equality.

To summarize; when given the task of determine whether the difference between two computed values AUC_1 and AUC_2 (where $AUC_1 > AUC_2$) is statistically insignificant (that the classifier corresponding to the second value is (at least) as good as the classifier corresponding to the first value), the hypothesis test with the test statistic Z and the hypotheses H_0 and H_1 described above will be employed. If H_0 is rejected under a small significance level, it is fairly certain that the difference is significant and we cannot conclude that the classifiers are equal. If H_0 is not rejected, we can only assume that H_0 is true and concludes that the difference is insignificant.

Genetic Algorithms

Often, optimization problems are search problems with very large search spaces. Genetic algorithms is a technique for finding near-optimal solutions for such problems. The first section of this chapter introduces optimization problems, while the rest of the chapter presents some aspects of genetic algorithms. See e.g. Goldberg (1989), Mitchell (1996), or Michalewicz (1996) for more comprehensive introductions.

5.1 Search as an Optimization Problem

All problems can be (re)formulated as search problems. The relevant aspects of the world at a given instance of time can be captured as the *state* of the world at that instance and a given problem can be solved by finding a path from the *initial state* representing the world as it is at the time before the problem is solved to a *goal state* representing a world where the problem is solved, using a set of *operators*. The set of operators (or the *successor function*) assign the directly reachable states to each state. Thus, a problem can be seen as a search through a (possibly infinite) state space $S = \{s_0, s_1, \dots, s_k, \dots\}$. Problems are often *solved* using additional information about the relative costs of the states and/or of the use of the different operators.

General search problems consists of finding the path from the initial state to the goal state, and a search algorithm is then a function mapping the problem – consisting of the state space with an initial state, a description of the goal states, a set of operators and possibly additional information – to a sequence of operator applications. Russell & Norvig (1995) view the operators as the actions available to an agent, and the solution to a problem is then the actions that must be performed in order to solve the problem. A search strategy is used to select the next operator to use in any given state. *Uninformed search methods* use strategies that are independent of the relative “goodness” of the states, examples are depth-first and breadth-first search. Such methods have poor performance – i.e. a great number of states must be visited before the goal state – for problems with large state spaces. In fact, these methods only enumerate the set of reachable states and visit them in order. *Informed search methods*, on the other hand, guide the search by using heuristics to evaluate the advantage of going to a particular state.

Some problems specify concrete goodness measures for each state in the state

space. This measure is not a heuristic, which is a “qualified guess” in the lack of a proper measure, but an actual evaluation of how useful a state is as a solution. A goodness measure is an evaluation function f :

$$f : S \rightarrow \mathbb{R}$$

The goal states for problems specifying a goodness measure are implicitly defined through the goodness measure and a threshold value specifying how “good” a solution must be to qualify as a goal state.

In addition, goal state is itself often a solution to the problem, and the path is not relevant. Such problems – specifying an evaluation function f and where a goal state itself is a complete solution – are *optimization problems*. The states that optimizes the evaluation function are sought. Since the path used to reach a state is irrelevant, so is the operators used. In an optimization problem, all states are in principle reachable from all states. Algorithms use operators to solve such problems, however, to guide the search. The difference compared to general search problems is that in the latter case operators are often a part of the problem, while for optimization problems the operators are specified by a particular algorithm. An algorithm that solves¹ optimization problems is shown in Figure 5.1. GENERIC-

```

function GENERIC-OPTIMIZATION( $S, f$ ) returns  $s \in S$ 
  inputs:
     $S$ , a state space
     $f : S \rightarrow \mathbb{R}$ , an evaluation function

  find the  $s \in S$  that maximizes  $f(s)$ 
  return  $s$ 
end function

```

Figure 5.1: A skeleton of an algorithm that solves optimization problems

OPTIMIZATION depends only upon the given state space and the evaluation function, and returns a single state. Implementations of algorithms that solve optimization problems differ in the operators they use to move from state to state. Optimization problems can be attacked by *iterative improvement methods*. These methods iteratively evaluate the current state, and select an operator to increase (not necessarily in the short run) the goodness measure. Consider the state space laid out in a two dimensional coordinate system, each coordinate corresponding to a state and each neighboring coordinate corresponding to a state reachable with an algorithm operator. Now let the goodness measure correspond to a third dimension. Then the state space can be seen as a landscape with the height at each point corresponding to the goodness measure of the state at that point. *Iterative improvement methods* try to find the highest peaks in the landscape. Each point in the landscape is a possible solution, and the methods try to iteratively improve the quality of the current (possible) solution. Since most of these methods usually only keep track of the current state and not a list of the already visited states and in addition do not look further ahead than the neighboring states, “this resembles trying to find the top of Mount Everest in a thick fog while suffering from amnesia” (Russell & Norvig 1995, p. 111). Nevertheless, traditional iterative improvement methods like hillclimbing and simulated annealing perform well on certain kinds

¹An algorithm can seldom be sure that there does not exist a better solution (without traversing the complete search space). The term “optimization” is thus abuse of language, but is very commonly used in the literature.

of problems. These methods are briefly introduced below. Genetic algorithms is another method that can be said to use an iterative improvement technique, although it has little in common to the other methods mentioned. This method is inspired by biological evolution, and iteratively updates a “population” of possible solutions by replacing the members of the population with new possible solutions that inherit properties from the best members of the old population.

5.1.1 Iterative Improvement Algorithms

Hillclimbing

Hillclimbing techniques always select the state that improves the goodness of the current state the most as the next state. That is, they always move in the direction of the steepest upward slope². Obviously, these algorithms are not good for solving optimization problems in general. The most important drawback is that the algorithms halt if they find local maxima³, that is, peaks that are lower than the highest peak. The success of a hillclimbing algorithm is thus highly dependent upon the starting point. A hillclimbing technique that tries to remedy this problem is *iterative*, or *random restart*, hillclimbing. This technique tries to find global maxima by running multiple iterations with different starting points.

Simulated Annealing

Instead of always moving up the steepest slope, simulated annealing techniques includes the possibility of taking a step downward to escape local maxima. When the algorithm reaches a peak, it can move to a lower state with a probability exponentially inversely proportional to the difference between the goodness of this state and of the current state. This probability is also positively dependent of the “temperature” of the system. As the temperature is gradually lowered, fewer and fewer changes are being made, until the algorithm terminates (at which point the temperature is zero and the algorithm behaves like a hillclimbing algorithm).

5.2 Informal Introduction to Genetic Algorithms

Algorithms loosely based on simulated evolution are often referred to as *evolution programs (EPs)*. An EP maintains a *population* of *individuals*, each individual representing a possible solution to a problem. Each individual has an associated *fitness*. The population is updated by generating a new *generation* of individuals on the basis of the current generation. The principles of evolution are used to create the new generation. For instance, parent individuals are selected stochastically according principle of natural selection (survival of the fittest), to be combined to create offspring individuals. Offspring individuals inherit the parents’ features (or *genes*). Another mechanism inspired by nature is mutation of single features – removal or addition of single genes. The mechanisms that are used to create new individuals from old individuals are called *genetic operators*.

²If the evaluation function is a cost function rather than a goodness function, these techniques are called *gradient descent techniques* and move in the direction of the steepest downward slope.

³Or local minima, in the case of gradient descent techniques.

Examples of natural selection in nature are plenty: an individual with “bad” genes (e.g. a bird with a very different color than that of the tree it lives in) has a higher probability of not surviving in its environment (e.g. being spotted by a predator). Thus, the probability that individuals with “good” genes will reproduce is higher than that of the individuals with “bad” genes, and the “good” genes (or good combinations of genes) have a higher chance of being carried on to the next generation (an example is a whole population of birds changing color over a number of generations). This mechanism is the motivation behind the use of EPs. By selecting the best individuals for reproduction, it is hoped that good combinations of genes will be found and that the average fitness of the population increases⁴.

Michalewicz (1996) defines as EPs all algorithms that are based on the principles of evolution and that maintains some kind of population, without any restriction on the representation of the individuals. Generally, EP individuals may have any representation, e.g. a tree, a DAG, a bitstring or a planning schedule. In the evolutionary programming literature the term *genetic algorithm* (GA) tends to be reserved for algorithms that represent individuals as fixed-length binary bitstrings. Michalewicz (1996, p. 3) defines (classical) GAs as:

- Using fixed-length binary bitstrings
- Using only the two binary genetic operators crossover and mutation

In this thesis the term genetic algorithm will denote an EP with fixed-length bitstring representation that only uses general evolution-inspired genetic operators (not necessarily limited to crossover and mutation). The more general class of EPs contain algorithms with problem-specific genetic operators. The main difference between a general EP and a GA is that to solve a problem using a EP, an algorithm attacking the particular problem is generally constructed, while to solve a problem using a GA, the problem must be reformulated to fit into a standard GA algorithm.

Genetic algorithms are commonly employed to solve optimization problems (see the previous section). They are massively parallel iterative improvement methods, where small genetic changes are performed on the the best individuals (i.e. states) in order to breed better individuals (move up the hill). The parallelism and the indeterminacy introduced by the common genetic operators ensures that the algorithm does not get stuck on local maxima.

5.3 Genetic Algorithms Defined

5.3.1 Concepts

Formal definitions of GA concepts are uncommon in the literature. In this section, general definitions of some concepts – as they will be used in this thesis – are presented. As a common terminology does not exist, the definitions of the following concepts are not necessarily shared by other authors.

The most important concept in a GA is the individual, which consists of a set of genes. A gene, or a *feature* or a *decoder*, g_i ($0 \leq i \leq m - 1$) is a member of a finite set $G = \{g_0, g_1, \dots, g_{m-1}\}$.

⁴EPs ignores the experiences (the relation between actions and rewards) learned by an individual in its lifetime; the evolution model is not Lamarckian.

Definition 5.1 (Individual, Chromosome Space). Given a set of genes G , the *chromosome space* over G is $U_G = P(G)$. A member $I \in U_G$ is called an *individual*, or a *chromosome*. \square

In classical GAs (Holland 1975), individuals are represented by strings of bits, where each “1” (“0”) indicates the presence (absence) of the particular gene in the individual corresponding to the position of the bit⁵. The string positions are called *loci*.

Definition 5.2 (String). Given a set of genes G , a *string*, or *bitstring*, B over U_G is a sequence $B = b_0b_1 \cdots b_{|G|-1}$ where $b_i \in \{0, 1\}$ ($0 \leq i \leq |G| - 1$). \square

Definition 5.3 (Semantics of a String). Given a set of genes G , the semantics of a string $B = b_0b_1 \cdots b_{|G|-1}$ over U_G is the individual $I = \llbracket B \rrbracket \in U_G$:

$$\llbracket b_0b_1 \cdots b_{|G|-1} \rrbracket = \{g_i | b_i = 1\}$$

The $\llbracket \cdot \rrbracket$ operator is isomorphic to a bijective function, and the corresponding inverse is denoted $\llbracket \cdot \rrbracket^{-1}$; e.g. $B = \llbracket I \rrbracket^{-1}$. \square

Given a set of genes G , a bitstring B over U_G thus uniquely identifies an individual $I \in U_G$. We will often refer to individuals by their corresponding bitstring, and sometimes – by abuse of language – talk about the individual B .

As mentioned in section 5.1, optimization problems use a goodness measure that operates on the states in the state space. In GAs, the goodness measure is called a fitness function.

Definition 5.4 (Fitness Function). A *fitness function* on a chromosome space U_G is a function:

$$fitness : U_G \rightarrow \mathbb{R}$$

\square

GAs operate on collections of individuals called populations.

Definition 5.5 (Population). A *population* P over a chromosome space U_G is a set of individuals

$$P \subseteq U_G$$

\square

A population is evolved into a new population by applying genetic operators to the members of the population.

Definition 5.6 (Genetic Operator). A *genetic operator* f of order k over a chromosome space U_G is a function

$$f : (U_G)^k \rightarrow (U_G)^k$$

\square

⁵In this chromosome model, each gene can be in one out of two states; “on” (present) or “off” (absent). In natural genetics, a gene can have more than two possible states called *alleles* or *feature values*.

The procedure used to apply a genetic operator to a set of individuals in a population is called a *genetic operation*. A genetic operation samples a set of parents and selects the particular genetic operators to be used from a class of operators. The two standard GA operators are mutation and crossover.

Definition 5.7 (Mutation). The first order genetic operator $m_p : U_G \rightarrow U_G$ where

$$m_p(\llbracket b_0 \cdots b_p \cdots b_{|G|-1} \rrbracket) = \llbracket b_0 \cdots \bar{b}_p \cdots b_{|G|-1} \rrbracket$$

where $0 \leq p \leq |G| - 1$, is a *mutation operator* over U_G . \square

Definition 5.8 (Crossover). The second order genetic operator $x_M : U_G \times U_G \rightarrow U_G \times U_G$ where

$$x_M(\llbracket B_1 \rrbracket, \llbracket B_2 \rrbracket) = (\llbracket B_1 \times M + B_2 \times \bar{M} \rrbracket, \llbracket B_1 \times \bar{M} + B_2 \times M \rrbracket)$$

for some $M \in \llbracket U_G \rrbracket^{-1}$, is a *crossover operator* over U_G . M is called the *crossover mask*, and for $I_a, I_b \in U_G$, the individuals in the pair $f(I_a, I_b)$ are called *offspring*. \square

(In the preceding definitions, \bar{B} denotes the binary inverse of a (possibly singular) bitstring B , and \times and $+$ the bitwise binary operations *and* and *or*, respectively). If the crossover mask is a continuous sequence of “1”s followed by a continuous sequence of “0”s, the operator is a *single point* crossover operator. Similarly, an operator with a mask consisting of a “1”s followed by “0”s and ending in a sequence of “1”s is a *two-point* crossover operator, and so on.

Application of these two types of genetic operators is illustrated by the next pair of examples.

Example 5.1 Consider the mutation operator m_3 applied to the individual

$$I_1 = \llbracket 11010011 \rrbracket$$

The result of the application is the individual

$$m_3(I_1) = \llbracket 11000011 \rrbracket$$

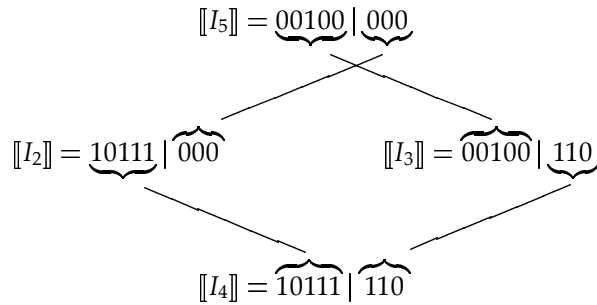
\diamond

Example 5.2 Consider the single point crossover operator $x_{11111000}$ applied to the individuals

$$I_2 = \llbracket 1011100 \rrbracket$$

$$I_3 = \llbracket 00100110 \rrbracket$$

The result of the application is the pair of individuals I_4 and I_5 :



where the bar (|) in the bitstrings denotes the single crossover point. \diamond

5.3.2 A Genetic Algorithm

An example of a generic genetic algorithm is shown in Figure 5.2.

function GENETIC-ALGORITHM(G , $fitness$, INIT(g), STOP($stats$), RESAMPLE(p), PARENTSELECT(p), UPDATESTATS(p,f), RECOMBINE(p,o), GENETICOPERATION₁(p), ..., GENETICOPERATION _{j} (p)) **returns** $Stats$, statistics

inputs:

G , a set of genes

$fitness : U_G \rightarrow \mathbb{R}$, a fitness function

INIT(g), a function returning a population $\subseteq U_G$

STOP($stats$), a stopping criteria

RESAMPLE(p), a function sampling a new population from an old

PARENTSELECT(p), a parent selection function

UPDATESTATS(p,f), a function updating statistics from a population and a set of fitness values

GENETICOPERATION _{i} (p) ($1 \leq i \leq j$), functions implementing genetic operations

RECOMBINE(p,o), a function recombining offspring with a population

$P \leftarrow \text{INIT}(G)$

for all $I_i \in P$ **do**

$F_i \leftarrow fitness(I_i)$

end for

$Stats \leftarrow \text{UPDATESTATS}(P,F)$

while not STOP($Stats$) **do**

$P \leftarrow \text{RESAMPLE}(P)$

$P_{par} \leftarrow \text{PARENTSELECT}(P)$

for $i := 1$ to j **do**

$Offspring \leftarrow \text{GENETICOPERATION}_i(P_{par})$

end for

$P \leftarrow \text{RECOMBINE}(P,Offspring)$

end while

return $Stats$

end function

Figure 5.2: A genetic algorithm

Implementations of GAs differ with respect to the input parameters to GENETIC-ALGORITHM. INIT generates the initial population. The algorithm updates statistics such as average and maximum fitness in each iteration, and its termination is determined by STOP. A new generation is generated from an old one by the RESAMPLE function which resamples the population. The resampling is generally done from a distribution that represents the fitness distribution of the individuals.

The functions GENETICOPERATION _{i} ($1 \leq i \leq j$) implement the genetic operations which sample from the parent population created by the function PARENTSELECTION. An individual in the parent population is selected by the genetic operation GENETICOPERATION _{i} with probability p_i . Let $size$ be the size of the population. A genetic operation:

1. Samples $p_i \cdot size$ k-tuples from the parent population

2. For each k-tuple; selects a genetic operator from a predefined set of genetic operators of order k and applies it to the tuple
3. Returns the set of offspring from the genetic operations

The selection of a genetic operator of a predefined type, e.g. two-point crossover operators (which corresponds to selection of the crossover points), is generally done randomly. The final recombination of the offspring into the population is done by the function RECOMBIMATE, for example by replacing the parents with the offspring.

5.4 Genetic Algorithms and Optimization Problems

Genetic algorithms are often used to solve optimization problems. Such problems consist of a state space S and a goodness function f , see figure 5.1. To use a GA to solve an optimization problem, the user must create a *coding* of the state space by constructing the following structures:

- G , the set of genes used to span out the chromosome space
- *meaning* : $U_G \rightarrow S$, a function used to map an individual into a state in the state space

Clearly, U_G is the GA's representation of S . The meaning of an individual, its *phenotype*, is of no concern to the GA. An algorithm solving a problem using a GA uses the function *meaning* to map from, for example, the best individual found by the GA to a (goal) state. The selection of a gene set G is arbitrary as long as the cardinality is large enough to represent S ($2^{|G|} \geq |S|$). In some problems, however, the states can be decomposed into genes in a very natural way. Figure 5.3 shows an algorithm that solves optimization problems using a GA. The function GET-

```

function GENETIC-OPTIMIZATION( $S, f, G, \text{meaning}$ ) returns  $s \in S$ 
  inputs:
     $S$ , a set of states
     $f : S \rightarrow \mathbb{R}$ , a goodness function
     $G$ , a set of genes
    meaning :  $U_G \rightarrow S$ , a mapping from individuals to states

   $Parameters \leftarrow \text{GETGENETICPARAMETERS}()$ 
   $I \leftarrow \text{FINDBESTINDIVIDUAL}(\text{GENETIC-ALGORITHM}(G, f \circ \text{meaning},$ 
     $Parameters))$ 
  return meaning( $I$ )
end function

```

Figure 5.3: Solving an optimization problem using a genetic algorithm

GENETICPARAMETERS() retrieves all but the three first parameters to GENETIC-ALGORITHM(...). FINDBESTINDIVIDUAL(stats) returns the best individual computed by the GA from a statistic structure. The fitness function used by the genetic algorithm maps an individual I into a real number r through the composition of the goodness function and the meaning function:

$$I \in U_G \xrightarrow{\text{meaning}} s \in S \xrightarrow{f} r \in \mathbb{R} \quad (5.1)$$

where $s \in S$.

5.5 Theoretical Foundations

Although a solid theoretical basis for genetic algorithms is lacking, some results exist and are commonly referenced in the literature. The aim of these results is to explain why GAs work, i.e. why they can be used to find iteratively better solutions.

Hypotheses trying to explain why GAs work often use the notion of a *schema*. Schemata are strings over the bitstring alphabet augmented with the “don’t-care” character “*”. A schema denotes the subset of the chromosome space (the solution space) containing all the individuals represented by bitstrings that matches the schema in all non-“*” positions. Schemata make it possible to talk about similarities of groups of individuals.

Definition 5.9 (Schema). Given a set of genes G , a *schema* S over G is a sequence $S = s_0s_1 \cdots s_{|G|-1}$ where $s_i \in \{0, 1, *\}$ ($0 \leq i \leq |G| - 1$). The *order* of a schema is

$$o(S) = |\{s_i | s_i \neq *\}|$$

The *defining length* of a schema is

$$\delta(S) = j - i$$

where i (j) is the smallest (largest) number such that $s_i \neq *$ ($s_j \neq *$)

□

Definition 5.10 (Semantics of a Schema). Given a set of genes G , the semantics of a schema $S = s_0s_1 \cdots s_{|G|-1}$ is the set of individuals $\{I_0, \dots, I_k\} = \llbracket S \rrbracket \subseteq U_G$:

$$\llbracket S \rrbracket = \{\llbracket B_0 \rrbracket, \dots, \llbracket B_k \rrbracket\}$$

where $\{B_0, \dots, B_k\}$ is the largest set of bitstrings such that, for all $B_j = b_{j_0} \cdots b_{j_{|G|-1}}$ and all i ($0 \leq i \leq |G| - 1$),

$$s_i = 0 \Leftrightarrow b_{j_i} = 0$$

$$s_i = 1 \Leftrightarrow b_{j_i} = 1$$

□

Consider the schema $S = *0*101**$. S represents a set of individuals containing e.g. 00010100 and 10010101 ($\llbracket 00010100 \rrbracket, \llbracket 10010101 \rrbracket \in \llbracket *0*101** \rrbracket$). The order of the schema is the number of non-“*” characters, $o(S) = 4$. The defining length is the difference in position between the last and the first non-“*” character in the schema, $\delta(S) = 4$.

The set of individuals in a population P *matched* by a schema S is $P \cap \llbracket S \rrbracket$. The *average fitness* of a schema S in a population P is the average fitness of the individuals in P matched by S . Under certain conditions, it can be concluded – by careful analysis of the number of individuals in a population matched by a schema S at a given time for “good” schemata S (see e.g. Michalewicz (1996)) – that the sampling rate of an above average schema increases with the number of iterations. This conclusion is called the *Schema Theorem*, or the *Fundamental Theorem of Genetic Algorithms*.

Hypothesis 4 (Schema Theorem) Schemata of small defining length, low order and above-average fitness receive exponentially increasing trials in subsequent generations of a genetic algorithm. \square

Thus, the more fit schemata will grow in influence in a population, especially if the non-***** characters are few and close together. Such schemata are called *building blocks*. The following hypothesis is often claimed to be valid.

Hypothesis 5 (Building Block Hypothesis) A genetic algorithm seeks near-optimal performance through the juxtaposition of short, low-order, high-performance schemata, called the building blocks. \square

The building block hypothesis describes the workings of the GA machinery like “a child [that] creates magnificent fortresses through the arrangement of simple blocks of wood” (Goldberg 1989, p. 41), and it has shown to hold on a wide range of empirical studies. The hypothesis is, however, highly controversial in the GA community. It builds on empirical studies rather than on formal proofs, and constructing examples that violates the hypothesis is not very difficult. Although widely cited, also the schema theorem fails to provide a proper theory of the workings of genetic algorithms in general. Thus, lacking a theoretical foundation, Hypotheses 4 and 5 should be used as explanations of why GAs often work in practice, rather than as proofs of formal properties.

Part II

Rule Filtering

CHAPTER 6

Rule Filtering

Hypothesis 1 (on page 3) states that a given complex model can be replaced with a model of a descriptive nature without lowering the predictive performance significantly. One strategy for finding a simple model with good classificatory capabilities is first inducing a general – possibly complex – model, and then pruning this model in a way that does not lower the performance significantly. Pruning can be done by identifying and removing the components of the model that only explain small parts of the data – e.g. random noise – thereby preserving the general trends in the underlying data material. In this chapter one particular method for pruning rule based models, called *rule filtering*, is introduced. Two concrete algorithms for rule filtering are then presented in the two following chapters. Empirical experiments employing these algorithms are documented in the last part of the thesis.

6.1 Model Pruning

In Chapter 1, the following properties of descriptive models were identified:

1. They are *not too complex* to be comprehended by humans
2. They exhibit *acceptable performance*, compared to more complex models

Here, *model pruning* is used to find such models. Model pruning involves post-pruning (see Section 1.3.1) an already induced model, in contrast to using pre-pruning techniques to create small models in the inductive learning step. Model pruning can thus be seen as an independent step used after the induction step in the knowledge discovery process, and can be used on any model regardless of the concrete algorithm used to induce the model.

Let MOD be a set of possible models for a phenomenon, and let the relation \preceq be a partial ordering on MOD . (MOD, \preceq) denotes a partial ordered set (poset). A model \mathcal{M}' in (MOD, \preceq) is called a *submodel* of a model \mathcal{M} in (MOD, \preceq) if $\mathcal{M}' \preceq \mathcal{M}$. In model pruning, submodels satisfying the two properties stated above are sought. To this end, two corresponding functions must be defined on the set of models. A *performance* function is used to compare the performance of the models:

$$performance : MOD \rightarrow [0, 1]$$

All performance functions discussed here are assumed to be normalized to the interval $[0, 1]$. A model \mathcal{M} with performance measure $performance(\mathcal{M}) = 1$ is a perfect model for the phenomenon¹. A *complexity* function is used to compare the complexity of the models:

$$complexity : MOD \rightarrow \mathbb{R}^+$$

The following is assumed to hold for a complexity function:

$$\mathcal{M}_2 \preceq \mathcal{M}_1 \Rightarrow complexity(\mathcal{M}_2) \leq complexity(\mathcal{M}_1) \quad (6.1)$$

$complexity(\mathcal{M})$ is a measure of the complexity of the model \mathcal{M} . We will be concerned with the *relative complexity* – the degree of complexity of a submodel $\mathcal{M}' \preceq \mathcal{M}$ compared to the model \mathcal{M} . The relative complexity is given by a function $complexity_{\mathcal{M}} : MOD \rightarrow [0, 1]$:

$$complexity_{\mathcal{M}}(\mathcal{M}') = \frac{complexity(\mathcal{M}')}{complexity(\mathcal{M})} \quad (6.2)$$

Clearly, model pruning is an optimization problem (see Section 5.1) with state space $S = MOD$. Given a model \mathcal{M} in (MOD, \preceq) , model pruning involves the following steps:

1. Defining the *complexity* and *performance* functions
2. Defining the evaluation function $f : MOD \rightarrow \mathbb{R}$ as a function of the complexity and the performance
3. Solving the optimization problem defined by MOD and f by traversing the search space $\{\mathcal{M}' | \mathcal{M}' \preceq \mathcal{M}\}$ to find the submodel \mathcal{M}' maximizing $f(\mathcal{M}')$.

6.2 Rule Filtering

The models under consideration in this thesis are sets of propositional decision rules (Chapter 3). How such sets can be used as classifiers was discussed in Section 3.6.

Rule filtering operates on an existing (potentially large) model, pruning it while retaining performance. Since the motivation for finding smaller models is that the models generated by the induction algorithms consist of a large number of rules (see Section 1.3), we suspect that individual rules can be removed from the large rule sets without lowering the performance of the resulting classifiers. Rule filtering attempts to find good submodels by removing whole rules.

In model pruning using rule filtering, the set of possible models, given a rule set RUL , is the power set of RUL : $MOD = P(RUL)$. Submodels are defined as subsets, the submodel relation \preceq equals the set inclusion relation \subseteq .

¹It may be the case that a perfect model does not exist, for example when the available information is insufficient to create a model with the desired resolution, or when the data collection procedure has been subjected to noise.

6.2.1 Complexity

The complexity of a rule based model RUL can be defined in several ways. Two of them are:

$$complexity_{card}(RUL) = |RUL| \quad (6.3)$$

$$complexity_{descr}(RUL) = \sum_{r \in RUL} \text{number of descriptors in } r \quad (6.4)$$

Both functions define complexity to be proportional to the number of “atomic” constituents in a model. They differ in regards to the resolution; in equation 6.3, single rules are considered to be the basic elements, in contrast to single descriptors in equation 6.4. The latter is the traditional measure of complexity of boolean formulae – complexity being equal to the number of boolean connectives (Pearl 1978) – over the language of decision rules.

Another, more general, possibility is to associate costs to the individual attributes in \mathcal{A} , and let the complexity function be a function of the costs of the attributes in the rules’ antecedents. This scheme is useful in domains where values for some attributes are more difficult or expensive to obtain than for others. Examples are results from different types of medical tests.

6.2.2 Performance

Methods for measuring the performance of classifiers were presented and discussed in Chapter 4. These methods measure the performance relative to a decision system. See Section 6.5 for some comments regarding optimizing the performance on a particular data set. Both the accuracy and the area under the ROC curve can be used as performance measures for rule filtering.

6.3 Evaluating Submodels

Based on the discussion in Chapter 4, we find that the area under the ROC curve is the proper measure for comparing classifier performance. $AUC_{\mathcal{A}}$ will henceforth be used as the performance measure relative to a decision system \mathcal{A} . In the following, it will be assumed that the decision system \mathcal{A} and the ROC focus class used in calculating the AUC is implicitly given. In an application of rule filtering, these will be set by the user.

Selecting a complexity function is not easy. The rule filtering process contains many other degrees of freedom, and the current investigation must be limited to one complexity function only. To this end, we have chosen to use the rule set cardinality as a measure of complexity instead of the more traditional complexity measure. The main reason for this is the idea for a strategy for limiting the search space of submodels if a “quality measure” can be defined on the level of the individual atomic constituents of a model. Several such quality measures are defined for individual rules. Further rationale is the experience that long rules in an induced rule set most often are due to noise in the originating decision system, and thus have low support counts. Therefore, long rules contributes little to the overall performance of a rule set, and will seldom be selected for inclusion in a filtered model if equation 6.3 is used. Using the rule count to evaluate complexity will thus not, in the general case, lead to models with high descriptor counts.

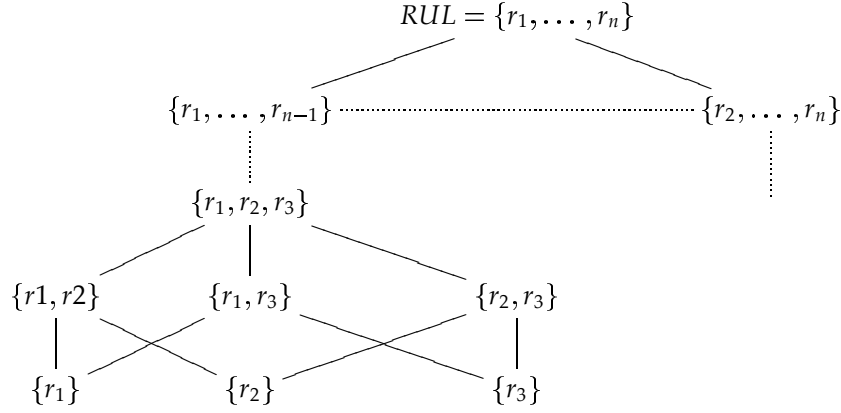


Figure 6.1: Hasse diagram of the lattice $(P(RUL), \subseteq)$.

The role of the evaluation function is to weight between the complexity and the acceptable performance of a submodel. The choice of a function combining the complexity and performance measures is user, application and domain dependent. In one application, for example, only models with five or fewer rules are interesting, no matter how good performance larger models might have. In another application, the evaluation criterion might be that the performance should be no lower than 99% of the performance of the unfiltered rule set, even if there exists considerably smaller models with performance equal to 98% of that of the unfiltered set. In the next chapters, two algorithms for finding good, small submodels are presented. The choice of the form of the evaluation function is discussed in connection with one of these algorithms. In the other algorithm, the evaluation criterion is implicitly defined in the algorithm itself, and needs no explicit definition.

6.4 Searching

Having defined an evaluation function over the submodels, model pruning is performed by searching the space of submodels for the best model according to the function. The cardinality of the search space for rule based models is:

$$|S| = |\{RUL' | RUL' \subseteq RUL\}| = 2^{|RUL|} \quad (6.5)$$

Clearly, the poset $(P(RUL), \subseteq)$ is a lattice². The search consists of traversing the lattice. The evaluation function can be tweaked to determine the optimal point(s) in the lattice. An outline of the submodel lattice for a rule set $RUL = \{r_1, \dots, r_n\}$ is shown in Figure 6.1. Each node in the diagram can be seen as being labeled with a complexity measure and a performance measure, and the node chosen by the search algorithm is determined by the evaluation function.

Typical induced rule sets, such as those computed by the ROSETTA system (Øhrn et al. 1998a), can have cardinalities with magnitudes of 100,000 rules. Clearly, the search space corresponding to such models are too large to be traversed by conventional search strategies, and other methods must be used.

²The least upper bound and greatest upper bound of two elements A and B of $(P(RUL), \subseteq)$ is $A \cup B$ and $A \cap B$, respectively.

6.5 Comments

The rule filtering problem was described as an optimization problem because the performance of the pruned models could be assessed. The problem is that a performance measure is accurate only for the data set used to compute it. The assessed performance tells only to what degree a theory fits the part of an equivalence relation represented by the data set, it can tell us nothing about how well the theory fits the complete goal relation. This problem is really implicit in inductive learning. Inductive learning is like telling the future; if we have access to an oracle that can verify the performance of the crystal ball, we do not need the crystal ball because we can use the oracle in the first place. Likewise, to assign a correct performance measure to a theory we would need access to the correct partition, and if we have access to the correct partition we need not construct the theory. Therefore, the assessed performance can never be more than an estimate. The question then is whether this estimate is good enough to be used to compare models, that is, if it should be used to define rule filtering as an optimization problem. Ideally, statistical tools should be used to give a better answer to whether the difference in estimated performance between two models are statistically significant. Unfortunately, the computational complexity and practical running times of the algorithms presented in the next chapters prevent anything but straightforward comparisons. It must be kept in mind that defining rule filtering as an optimization problem is a simplification, and that the algorithms actually only optimizes the performance on a hold-out set. If both the hold-out set and the testing set represent the “real” distribution of data in the particular domain, the models with high estimated performance on the hold-out set will also have high estimated performance on the testing set.

Quality-Based Filtering

7.1 Introduction

Generally, the cardinality of the search space corresponding to the rule filtering optimization problem is extremely large. One solution to this problem is employing special search algorithms, such as genetic algorithms. This possibility is explored in the next chapter. In this chapter, an algorithm that only explores a very small, well-defined fraction of the search space is investigated. The motivation was stated in Hypotheses 2 and 3 on page 4. Hypothesis 2 states that there exist a function creating an ordering of the rules in a rule set, such that a minimal model satisfying a given performance constraint can be obtained by successively including the rules with the best quality measures until the constraint is satisfied. A great number of small models (subsets of a rule set) generally exists. Hypothesis 3 on page 4 states that the models consisting of the “best” individual rules in the ordering also are the overall best models. The function creating the ordering of the rules will be called a *quality function*, and will be defined as assigning a “quality” measure to each rule. Rule quality is, in addition to overall model performance, another evaluation measure, but on the level of single rules instead of sets of rules. The hypothesis effectively states that the performance of a model is a function of the performance (quality) of the single rules. According to this hypothesis, for a given model performance constraint, the best submodel is unambiguously defined by iteratively increasing the threshold and removing rules until no more rules can be removed without violating the model performance constraint. If Hypothesis 3 holds, the rule filtering problem can be easily solved since only at most one model of a given complexity needs to be considered. In the following section, several formulas for measuring quality are presented.

7.2 Defining Rule Quality

The rule quality for the rules in a rule set RUL is determined by a *quality* function:

$$quality : RUL \rightarrow \mathbb{R} \quad (7.1)$$

Defining a quality function consists of identifying the properties of “good” rules. The function is defined so that is maximized for certain properties. Two properties commonly used to define “good” rules is *accuracy* and *coverage* (see Definitions 3.7

and 3.8 on page 20). The first property is a measure of how reliable – or accurate – a rule is, the second is a measure of how important – or powerful – a rule is. Generally, we would like rules to be both accurate and powerful. The question is how these two measures should be combined to one numerical overall quality measure. The literature is rich with functions combining the accuracy and coverage properties, as well as other properties calculated from the contingency table (Section 3.4), for various purposes. Bruha (1997) summarizes some of the approaches¹. In the following, different contingency table based rule quality formulas are presented.

7.2.1 Rule Quality Formulas

The formulas given below are relative to the contingency table for the given rule. $|U|$ refers to the cardinality of the originating information system, i.e. the total sum in the contingency table.

Michalski

Michalski (1983) suggests that high accuracy and coverage are requirements of decision rules. Bergadeno et al. (1988) combine these measures into a quality measure similar to the following function:

$$quality_{Michalski}(r) = \mu \cdot accuracy(r) + (1 - \mu) \cdot coverage(r) \quad (7.2)$$

$quality_{Michalski}$ is a weighted sum of the measures of the accuracy and the coverage properties; the weight μ will henceforth be called the *accuracy bias*. The accuracy bias is typically user defined, and is generally selected on an experimental basis. Thus, using equation 7.2 can be seen as an empirical approach without any theoretical foundation.

Torgo

One variation of equation 7.2, is the accuracy bias employed by Torgo (1993):

$$\mu = \frac{1}{2} + \frac{1}{4}accuracy(r) \quad (7.3)$$

In the above equation, μ is not fixed but a function of the accuracy. The formula is based on experiments by Torgo on real world domains, and the accuracy value is judged to be the most important of the two properties in equation 7.2.

Brazdil

Brazdil & Torgo (1990) define a quality function as a product of the accuracy and coverage measures instead of a sum:

$$quality_{Brazdil}(r) = accuracy(r) \cdot e^{coverage(r)-1} \quad (7.4)$$

Similar to the Michalski and Torgo functions, $quality_{Brazdil}$ is based on empirical experiments, rather than on theoretical foundations.

¹ Actually in the context of classifying schemes for unordered rule sets, as briefly discussed in Section 3.6.

Pearson χ^2 statistic

Bishop et al. (1991) describe several measures related to the contingency table. Bruha (1997) has found that two of these statistics may be used as measures of rule quality. One of the statistics is the *Pearson χ^2 statistic*:

$$quality_{\chi^2}(r) = \frac{n_{\alpha,\beta} \cdot n_{\neg\alpha,\neg\beta} - n_{\alpha,\neg\beta} \cdot n_{\neg\alpha,\beta}}{n_{\beta} \cdot n_{\neg\beta} \cdot n_{\alpha} \cdot n_{\neg\alpha}} \quad (7.5)$$

The statistic is χ^2 distributed with one degree of freedom.

G2-likelihood statistic

The other statistic described by Bishop et al. (1991) and recommended by Bruha (1997), also χ^2 distributed with one degree of freedom, is the *G2-likelihood statistic*:

$$quality_{G2}(r) = 2 \left(n_{\alpha,\beta} \ln \frac{n_{\alpha,\beta} \cdot |U|}{n_{\alpha} \cdot n_{\beta}} + n_{\alpha,\neg\beta} \ln \frac{n_{\alpha,\neg\beta} \cdot |U|}{n_{\alpha} \cdot n_{\neg\beta}} \right) \quad (7.6)$$

The G2-likelihood statistic divided by $2|U|$ is equal to the *J-measure* (Smyth & Goodman 1990):

$$quality_J(r) = \frac{quality_{G2}(r)}{2|U|} \quad (7.7)$$

Cohen

Cohen (1960) presents a measure from the contingency table, interpreted as a rule quality measure by Bruha (1997):

$$quality_{Cohen}(r) = \frac{|U| \cdot n_{\alpha,\beta} + |U| \cdot n_{\neg\alpha,\neg\beta} - n_{\alpha} \cdot n_{\beta}}{|U|^2 - n_{\alpha} \cdot n_{\beta} - n_{\neg\alpha} \cdot n_{\neg\beta}} \quad (7.8)$$

Equation 7.8 is measure of the sum $f_{\alpha,\beta} + f_{\neg\alpha,\neg\beta}$ in the frequency based contingency table, corrected by the sum $f_{\alpha} \cdot f_{\beta} + f_{\neg\alpha} \cdot f_{\neg\beta}$ and normalized. A large value for the sum of $f_{\alpha,\beta}$ and $f_{\neg\alpha,\neg\beta}$ is clearly a desired property of a decision rule.

Coleman

Bishop et al. (1991) define a measure, called the *Coleman function*, similar to the Cohen formula:

$$quality_{Coleman}(r) = \frac{|U| \cdot n_{\alpha,\beta} - n_{\alpha} \cdot n_{\beta}}{n_{\alpha} \cdot n_{\neg\beta}} \quad (7.9)$$

Equation 7.9 only considers the first element $f_{\alpha,\beta}$ in the main diagonal, instead of both elements as in equation 7.8. The Coleman quality function exhibits high quality for a rule if many of the objects covered by the antecedent also are covered by the consequent. Unlike the Cohen function, it does not reward agreement between the number of objects *not* covered by the antecedent and the consequent, respectively. The Coleman function thus gives a measure of the accuracy², while the

²However, the Coleman function is not identical to the *accuracy* function; $quality_{Coleman}(r) = \frac{accuracy(r) - f_{\beta}}{1 - f_{\beta}}$.

Cohen function also includes a measure of the coverage. Bruha (1997) proposes the two following modifications of the Coleman function to introduce coverage dependence:

$$quality_{C1}(r) = quality_{Coleman}(r) \frac{2 + quality_{Cohen}(r)}{3} \quad (7.10)$$

$$quality_{C2}(r) = quality_{Coleman}(r) \frac{1 + coverage(r)}{2} \quad (7.11)$$

Kononenko

The following function is an information theoretical measure of rule quality, proposed by Kononenko & Bratko (1991):

$$quality_{Kononenko}(r) = -\log_2 f_\beta + \log_2 accuracy(r) \quad (7.12)$$

7.2.2 Discussion

The Michalski, Torgo and Brazdil formulas are empirical, ad-hoc formulas, but are intuitive and interpretable. The rest of the formulas are based on theoretical foundations, and may be more difficult to interpret directly. Pearson, G2/J, Cohen and Coleman/C1/C2 are based on statistics and the theory of contingency tables. The two former formulas give *association* measures for the contingency table, while the two latter give *agreement* measures (see Bishop et al. (1991)). The Kononenko formula is based on information theory.

A summary of the rule quality formulas is given in Table 7.1. The table shows the domain and range for each function. The domain for a function is considered to be all possible rules modulo some assumptions given below, except those with contingency matrices corresponding to the “Undefined when” column. The ranges are not given relative to a particular decision system. For example, the range for the Coleman formula is $[-(|U| - 1), 1]$ for a particular decision system.

Function Domains

It is assumed that $n_\alpha > 0$ (that a rule covers at least some of the objects) and that both $n_\beta > 0$ and $n_{\neg\beta} > 0$ (that there exist objects of both classes for a binary decision attribute). Rules violating the first assumption make no sense, and generating decision rules for decision systems violating the second assumption is superfluous. These assumptions are also necessary for the rule accuracy and coverage to be well-defined. An additional assumption is made regarding the association based formulas. These formulas are only valid if the class indicated by β is the *dominating* decision class for the rule, that is, if $n_{\alpha,\beta} > n_{\alpha,\neg\beta}$. The implementation of the algorithms will ensure that this assumption is not violated, see Section A.4 in the appendices.

However, some of the formulas are undefined even for certain contingency matrices and decision tables not violating the above assumptions, as given by the “Undefined when” column in Table 7.1. For example, the Pearson formula is undefined when the rule covers *all* the objects in the decision system, and the G2 and J formulas are undefined if the rule classifies all objects correctly.

Quality measure	Formula	Range	Undefined when
Michalski	$\mu \times accuracy(r) + (1 - \mu) \times coverage(r)$	$[0, 1]$	
Torgo	Michalski with $\mu = \frac{1}{2} + \frac{1}{4}accuracy(r)$	$[0, 1]$	
Brazdil	$accuracy(r) \cdot e^{coverage(r)-1}$	$[0, 1]$	
Pearson	$(n_{\alpha,\beta} \cdot n_{\neg\alpha,\neg\beta} - n_{\alpha,\neg\beta} \cdot n_{\neg\alpha,\beta}) / (n_{\beta} \cdot n_{\neg\beta} \cdot n_{\alpha} \cdot n_{\neg\alpha})$	$[0, 1]$	$n_{\neg\alpha} = 0$
G2	$2(n_{\alpha,\beta} \ln(n_{\alpha,\beta} \cdot U / (n_{\alpha} \cdot n_{\beta})) + n_{\alpha,\neg\beta} \ln(n_{\alpha,\neg\beta} \cdot U / (n_{\alpha} \cdot n_{\neg\beta})))$	$[0, \infty]$	$n_{\alpha,\neg\beta} = 0$
J	$quality_{G2}(r) / (2 U)$	$[0, \infty]$	$n_{\alpha,\neg\beta} = 0$
Cohen	$(U \cdot n_{\alpha,\beta} + U \cdot n_{\neg\alpha,\neg\beta} - n_{\alpha} \cdot n_{\beta}) / (U ^2 - n_{\alpha} \cdot n_{\beta} - n_{\neg\alpha} \cdot n_{\neg\beta})$	$[-1, 1]$	$ U ^2 = n_{\alpha}n_{\beta} + n_{\neg\alpha}n_{\neg\beta}$
Coleman	$(U \cdot n_{\alpha,\beta} - n_{\alpha} \cdot n_{\beta}) / (n_{\alpha} \cdot n_{\neg\beta})$	$[-\infty, 1]$	
C1	$quality_{Coleman}(r) / (2 + quality_{Cohen}(r)) / 3$	$[-\infty, 1]$	$ U ^2 = n_{\alpha}n_{\beta} + n_{\neg\alpha}n_{\neg\beta}$
C2	$quality_{Coleman}(r) / (1 + coverage(r)) / 2$	$[-\infty, 1]$	
Kononenko	$-\log_2 f_{\beta} + \log_2 accuracy(r)$	$[-\infty, 0]$	$accuracy(r) \geq f_{\beta}$

Table 7.1: Summary of quality functions

Available Results

Bruha (1997) has tested the applicability of the above quality measures with different classification schemes. The investigation indicates that all the measures may be monotone towards the accuracy and coverage, but this cannot be proved for all the formulas. The experiments show that all the tested measures where applicable for the particular task, and that the empirical measures exhibit quite good performance in spite of the lack of theoretical foundations. The statistical measures were found to perform somewhat better, but no statistical significance test is used to detect any significant difference between the empirical and the statistical measures. The Michalski formula generally showed better performance with higher accuracy weights.

7.3 Quality-Based Rule Filtering

Having defined a quality function *quality*, filtering a rule set *RUL* can be done by selecting a submodel

$$RUL' \in \{\{r \in RUL \mid \text{quality}(r) \leq t\} \mid t \in \mathbb{R}\}$$

As a matter of convenience, a rule set corresponding to the particular threshold value t above will be denoted RUL_t . Observe that the set of submodels corresponding to threshold values for a given quality function not necessarily contains a rule set with a particular cardinality $|RUL'| < |RUL|$, since the quality function can map two or more rules to the same quality value.

The function `QUALITYTHRESHOLD-PERFORMANCE-GRAPH` in Figure 7.1 creates a graph from a subset of \mathbb{R} to the interval $[0, 1]$, corresponding to the estimated performance of the rule set RUL_t as a function of the threshold value t . The minimum number of rules included for each threshold value is determined by `RESOLUTION(k)`, where k is the rule count for the previous threshold value. Resolution is further discussed in Section 7.3.3. The actual rule pruning can be performed by selecting a threshold, and thus the corresponding rule set RUL_t , from the graph. This simple algorithm is shown in Figure 7.2.

Example 7.1 (Quality-Based Rule Filtering) The rule set *RUL* is to be filtered using the rule quality function *quality*. The workings of `QUALITYTHRESHOLD-PERFORMANCE-GRAPH` is illustrated in Table 7.2. The table shows the rules in

r_k	$t = \text{quality}(r_k)$	$\text{performance}(RUL_t)$
r_1	0.92	0.56
r_2	0.87	0.74
r_3	0.87	0.74
r_4	0.83	0.75
r_5	0.79	0.81
...

Table 7.2: The sorted rules in `QUALITYTHRESHOLD-PERFORMANCE-GRAPH`

RUL sorted by quality, and the classifier performance corresponding to the accumulated rule set. For example, $RUL_{0.87} = \{r_1, r_2, r_3\}$. The graph computed

function QUALITYTHRESHOLD-PERFORMANCE-GRAPH(RUL , $performance$, $quality$) **returns** $\{(t, p)\} \subseteq \mathbb{R} \times [0, 1]$

inputs:
 RUL , a set of rules
 $performance : P(RUL) \rightarrow [0, 1]$, a performance function
 $quality : RUL \rightarrow \mathbb{R}$, a rule quality measure

Sort the rules in RUL according to $quality$. Let the tuple $(r_1, \dots, r_{|RUL|})$ denote the sorted rules, where $quality(r_i) \geq quality(r_j)$ if $i \leq j$.

$k \leftarrow 1$
 $G \leftarrow \emptyset$
while $k \leq |RUL|$ **do**
 $RUL' \leftarrow \emptyset$
 $m \leftarrow 0$
 $n \leftarrow \text{RESOLUTION}(k)$
 repeat
 $t \leftarrow quality(r_k)$
 repeat
 $RUL' \leftarrow RUL' \cup \{r_k\}$
 $m \leftarrow m + 1$
 until $quality(r_k) \neq t$ or $k > |RUL|$
 until $m = n$ or $k > |RUL|$
 $k \leftarrow k + 1$
 $p \leftarrow performance(RUL')$
 $G \leftarrow G \cup \{(t, p)\}$
end while
return G
end function

Figure 7.1: The quality-based rule filtering algorithm

function QUALITYBASED-RULEPRUNING(RUL , $quality$, t) **returns** $RUL' \subseteq RUL$

inputs:
 RUL , a set of rules
 $quality : RUL \rightarrow \mathbb{R}$, a rule quality measure
 $t \in \mathbb{R}$, a quality threshold

Sort the rules in RUL according to $quality$. Let the tuple $(r_1, \dots, r_{|RUL|})$ denote the sorted rules, where $quality(r_i) \geq quality(r_j)$ if $i \leq j$.

$RUL' \leftarrow \{r_i | quality(r_i) \leq t\}$
return RUL'
end function

Figure 7.2: Rule Filtering using quality threshold

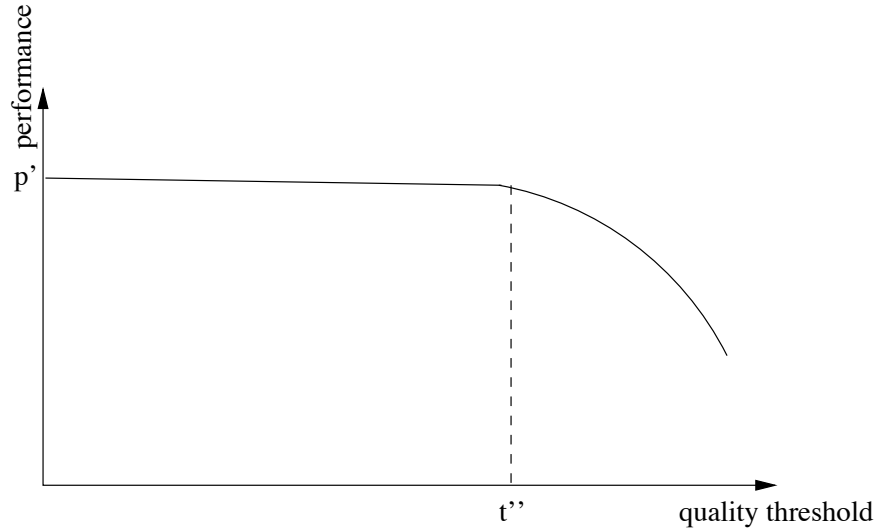


Figure 7.3: Sample performance vs. threshold graph

by QUALITYTHRESHOLD-PERFORMANCE-GRAPH is shown in Figure 7.3. Here, $performance(RUL) \equiv p'$. One choice of a submodel is the rule set $RUL_{t''}$ corresponding to the threshold t'' at the knee-point of the curve. \diamond

The approach illustrated by Example 7.1 is taken by Øhrn et al. (1998b). In this study, a real-world medical data set consisting of data regarding 5625 patients is used for learning, pruning and testing. Induced rule sets, with more than 30,000 rules, are pruned by filtering according to *coverage* threshold (identical to using the Michalski quality function with $\mu \equiv 0$). Thresholds are selected by visual inspection of the graphs. The pruned rule sets contain between 5 and 15 rules, and, apparently, show little reduction in performance compared to the full sets. No alternative quality measures are investigated in this study.

7.3.1 Model Selection

The graph produced by the filtering algorithm contains a range of models of varying complexity and performance. The *shape* of a graph generated by a quality formula reflects the formula's suitability for filtering purposes. To be useful for quality filtering, the performance function must exhibit monotony towards the quality threshold. In other words, the graph should be non-increasing towards increasing threshold value. A graph with a near-perfect shape is shown in Figure 7.3. Perfect behavior is, of course, difficult to achieve with real world data, and small ripple in the graphs must be tolerated.

In practical applications, a method for selecting a particular model is needed. Due to the many varying properties of graphs created from real-world data (see Chapters 9 and 10), it is difficult to formally define a set of model preference rules. The selection of the models necessarily must include an element of subjectivity. For graphs with “good” shape, the selection criteria could include finding the start of the knee-point or finding the point where the difference in performance, compared to the unfiltered rule set, becomes statistically significant.

7.3.2 Undefined Quality

As discussed in Section 7.2.2, not all quality measures are defined for all possible rules. Some formulas are undefined only in obscure cases, while others are undefined in cases normally associated with high quality. Examples of the latter are the G2/J measures, which are undefined when the rule only covers objects of the correct class – that is, for rules with perfect accuracy ($accuracy(r) = 1$). Intuitively, such rules should be considered to be “good”. In the implementation of the interface to the algorithms in Figures 7.1 and 7.2 the user can select between the following ways of handling undefined quality values:

1. The sorting algorithm can assign a special quality value \top to rules with undefined quality. \top is larger than any other quality value. Rules with the quality value \top have equal quality.
2. The interface can remove rules with undefined quality before passing the rules to the algorithms in Figures 7.1 and 7.2.

In the first case, all rules with undefined quality are given an equal quality better than all other rules. This may lead to a great number of rules with equal best quality, and may render the quality filtering strategy useless if the goal is small rule sets. The second case avoids this problem, but does not utilize potentially good rules.

7.3.3 Resolution

The function RESOLUTION in Figure 7.1 determines the resolution of the computed graph. The following variants have been implemented:

1. *Fixed resolution*

$$RESOLUTION(k) = c$$

where c is a constant ($1 \leq c \leq |RUL|$).

2. *Dynamic resolution*

$$RESOLUTION(k) = \begin{cases} pk & \text{if } pk < max \\ max & \text{otherwise} \end{cases}$$

where $0 \leq p \leq 1$.

Fixed resolution is used to include at least a constant number of rules for each threshold value. Maximum resolution (the number of rules for each threshold equal to the number of rules having with quality measure equal to that threshold) is achieved using fixed resolution with $c = 1$. With dynamic resolution, the number of rules (at least) included for each threshold value is a fraction of the number of accumulated rules, upward bounded by a constant max . Dynamic resolution is typically useful when filtering large rule sets with few rules having equal quality measures. Filtering such rule sets with a small fixed resolution may be time consuming. If only the smallest subsets are of interest, dynamic resolution can be used to decrease the number of evaluations needed for the largest subsets.

7.4 The Search Space

In quality-based rule filtering, at most one model with a particular number of rules is considered. The cardinality of the search space therefore has an upper bound $|RUL|$, where RUL is the unfiltered rule set. The cardinality is less than $|RUL|$ if two or more rules have identical quality values. In other words, the cardinality of a submodel is uniquely defined by the threshold value, but a submodel is not necessarily defined for a particular cardinality. A plot of performance versus cardinality will thus be equal to the performance versus threshold plot.

Strictly speaking, model pruning – and rule filtering – have been defined as optimization problems (Section 6.1). However, optimization of a model evaluation criteria is not mentioned in the above discussion. The reason is that often when doing quality-based rule pruning, an evaluation function is not defined explicitly but rather implicitly in the procedure used for selecting the pruned model (such as selecting the knee-point in Example 7.1).

Of course, quality-based filtering using an evaluation criterion directly can easily be done. This approach is shown in Figure 7.4.

```

function    QUALITYBASED-EVALUATION-RULEPRUNING( $RUL$ ,     $quality$ ,
 $performance$ ,  $f$ ) returns  $RUL' \subseteq RUL$ 
  inputs:
     $RUL$ , a set of rules
     $performance : P(RUL) \rightarrow [0, 1]$ , a performance function
     $quality : RUL \rightarrow \mathbb{R}$ , a rule quality measure
     $f : P(RUL) \rightarrow \mathbb{R}$ , an evaluation function

   $RUL_{best} \leftarrow \emptyset$ 
   $G \leftarrow \text{QUALITYTHRESHOLD-PERFORMANCE-GRAPH}(RUL, \text{performance},$ 
 $quality)$ 
   $T \leftarrow \{t \mid (p, t) \in G\}$ 
  for all  $t \in T$  do
     $RUL' \leftarrow \text{QUALITYBASED-RULEPRUNING}(RUL, \text{quality}, t)$ 
    if  $f(RUL') > f(RUL_{best})$  then
       $RUL_{best} \leftarrow RUL'$ 
    end if
  end for
  return  $RUL_{best}$ 
end function

```

Figure 7.4: Rule Filtering using an evaluation function

CHAPTER 8

Genetic Filtering

8.1 Introduction

Because rule filtering (see Chapter 6) is an optimization problem, it can be attacked by using a genetic algorithm (see Chapter 5). When using a genetic algorithm for rule filtering, the state space is investigated in a parallel search and evaluation is performed on complete models (in contrast to single rules) only. The rule filtering problem is well suited for genetic algorithms. If a natural coding with the rules in the original rule set as the set of genes is used, every possible individual is a potential solution. That is, there does not exist any invalid combination of genes. General advantages of genetic algorithms are that they are highly parallel and have shown outstanding results in some domains. The parallelism is important in the rule filtering problem. A search space for this problem typically has many local maxima, and traditional hillclimbing techniques will not perform very well. As an example consider removing single rules from a rule set until the performance drops beyond a certain point, a hillclimbing technique. This algorithm will stop when no single rule can be removed without violating the constraint, even if the removal of two rules in succession actually would increase the performance. An important disadvantage of using genetic algorithms is the poor theoretical basis.

In the next section, the coding of the rule filtering optimization problem is presented, followed by a discussion on the fitness function. In the last section, the genetic algorithm skeleton defined in Chapter 5 is filled in. In the rest of the chapter, the rule set given for pruning will be called *RUL*.

8.2 Coding

As discussed in Chapter 6, model pruning is an optimization problem defined by the search space $S = MOD$ and the evaluation function f . To solve this optimization problem using a genetic algorithm, a coding of the state space must be constructed (see Section 5.4).

Fortunately, the rule filtering problem lends itself to a natural coding. Here $S = MOD = P(RUL)$, for a given rule set *RUL*. We will code submodels as individuals

by using the individual rules as genes:

$$G = RUL \quad (8.1)$$

With this natural coding, $U_G = P(RUL) = S$, and the meaning function on the chromosome space $meaning : U_G \rightarrow S$ (Section 5.4) is the identity function $\iota_{P(RUL)}$.

8.3 The Fitness Function

When solving an optimization problem with a genetic algorithm, an evaluation function f and a meaning function $meaning$ (see equation 5.1) mapping an individual into a real number, must be defined (see Section 5.4). The fitness function (Definition 5.4) in the genetic algorithm is the composition of the evaluation function and the meaning function. In the current problem where $meaning$ is the identity function, the fitness function is equal to the evaluation function. Below, this function will be referred to as the fitness function for the genetic algorithm, but formally it is the evaluation function for the optimization problem. For simplicity, the following discussion will be in terms of the fitness function. The reader should keep in mind that, in the rule filtering problem, this is equal to defining the general evaluation function for the optimization problem, and the following discussion is not exclusively related to genetic algorithms.

8.3.1 General Form

Given a rule set RUL , the fitness function

$$fitness : P(RUL) \rightarrow \mathbb{R} \quad (8.2)$$

creates a landscape of the search space. In the rule filtering problem, the evaluation function – and hence the fitness function – is a function of two variables; the complexity measure and the performance measure. In GA problems with two independent variables, fitness functions on the form $f = a \cdot x + b \cdot y$ are commonly used. For the rule filtering problem, the following general form of the basic fitness function is chosen:

$$fitness_\lambda(RUL') = \lambda performance(RUL') + (1 - \lambda)(1 - complexity_{RUL}(RUL')) \quad (8.3)$$

where $RUL' \subseteq RUL$ and $0 \leq \lambda \leq 1$. The factor λ , henceforth called the *performance bias*, reflects the relative importance of performance versus complexity of the possible solutions. The fitness function is inversely dependent on the relative complexity measure, that is, the function is maximized by rule sets with high performance and low complexity. The solid line curve in Figure 8.1 is a plot of fitness value versus complexity for a fixed performance measure.

8.3.2 Selecting a Performance Bias

Section 6.3 mentioned the diversity of model evaluation criteria for different applications. Here, tailoring the fitness function in Equation 8.1 to a particular application is discussed.

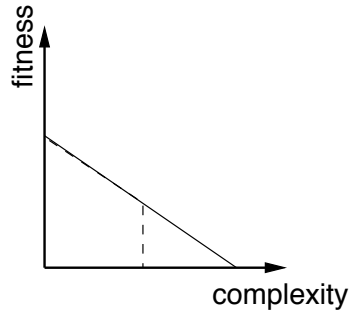


Figure 8.1: The fitness functions $fitness_{\lambda}$ (solid line) and $fitness_{\lambda,n}$ (dashed line), with fixed performance. The gradient is $-(1 - \lambda)$.

Selecting the performance bias in practice amounts to deciding which models should be considered *equivalent* with respect to the evaluation criterion. When pruning a given model, how large decrease in performance is acceptable relative to the decrease in complexity, in order to say that the pruned model has the same value as the original model? Consider the fitness function as a function of the two factors in equation 8.3: $f(p, c) = \lambda \cdot p + (1 - \lambda) \cdot c$ (where p is a performance measure and c is one minus a relative complexity measure). A class of equivalent models is characterized by a constant fitness value k :

$$\lambda \cdot p + (1 - \lambda) \cdot c = k \quad (8.4)$$

The factor c can be expressed as a function of p :

$$\begin{aligned} c(p) &= \frac{k - \lambda \cdot p}{1 - \lambda} \\ &= \frac{k}{1 - \lambda} - \frac{\lambda}{1 - \lambda} \cdot p \\ &= a - b \cdot p \end{aligned} \quad (8.5)$$

Thus, the equivalence between models can be described by a class of straight lines, henceforth called *equivalence curves*, in the (p, c) plane, each class corresponding to one value of the performance bias λ . The constant a in Equation 8.5 determines the point $c(0)$ for a given curve. The constant b is solely defined by a particular λ and determines the gradient of the lines. A class of equivalence curves for a fixed λ is then solely characterized by the gradient. This is illustrated in Figure 8.2. Selecting a performance bias is equal to selecting a class of equivalence curves, i.e. a gradient. Since the curves are straight lines, this can be done by selecting two points (p_1, c_1) and (p_2, c_2) on one of the curves. On an equivalence curve, the fitness at all points is equal:

$$\begin{aligned} f(p_1, c_1) &= f(p_2, c_2) \\ \lambda \cdot p_1 + (1 - \lambda) \cdot c_1 &= \lambda \cdot p_2 + (1 - \lambda) \cdot c_2 \\ \lambda &= \frac{c_2 - c_1}{c_2 - c_1 + p_1 - p_2} \end{aligned}$$

or, letting $\Delta x = x_2 - x_1$,

$$\lambda = \frac{\Delta c}{\Delta c - \Delta p} \quad (8.6)$$

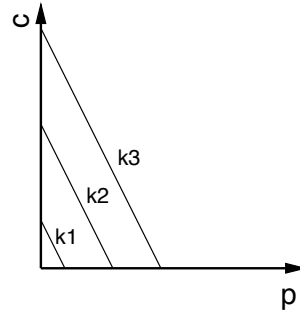


Figure 8.2: Complexity versus performance for fixed λ . Each line corresponds to a fitness value k_i . The gradient is $-\lambda/(1 - \lambda)$.

Equation 8.6 can be used in practice by selecting two models that considered to be equally desirable from the application viewpoint.

Example 8.1 (Setting the Performance Bias) Consider a model RUL with $|RUL| = 100,000$. If we are looking for small models, we might consider a decrease in the number of rules from 15 to 10 to be worth a decrease in performance from 0.90 to 0.85, giving:

$$\Delta c = 1 - \frac{10}{100,000} - \left(1 - \frac{15}{100,000}\right) = 0.00005$$

$$\Delta p = 0.85 - 0.90 = -0.05$$

Using equation 8.6,

$$\lambda = \frac{0.00005}{0.00005 + 0.05} \approx 0.001$$

◇

In using equation 8.6, it is very important to choose the “right” points to determine the Δ ’s. The right points correspond to models that are realistic candidates to be accepted as the final product of the pruning process. It would be obscure, for example, to use the point corresponding to the start model RUL as point (p_1, c_1) in example 8.1, because of the extreme complexity compared to that of acceptable models. It must be assumed that models with significantly less complexity and comparable (or better) performance exist. Actually, if large models are irrelevant, they need not be considered at all. This is discussed in the next section.

8.3.3 Using a Cutoff-Value

In some applications, only very small models are of interest. For example, filtering away 99% of a rule set with 100,000 rules might not be of any value at all because a model with 1000 rules may still be too complex to be comprehended by humans. Furthermore, with no restriction on complexity, optimal settings for the genetic algorithm are extremely hard to find, and even if these are found the algorithm might converge very slowly towards small enough models¹. The models

¹The point is that it takes a long time before interesting solutions are found, not that the algorithm does not converge early, which generally is a good feature in a GA problem.

of interest in this work, i.e. models of a descriptive nature, are exactly very small models. Therefore, the fitness function is refined to severely punish models with complexity outside of a predefined range:

$$fitness_{\lambda,n}(RUL') = \begin{cases} fitness_{\lambda}(RUL') & \text{if } complexity(RUL') < n \\ 0 & \text{otherwise} \end{cases} \quad (8.7)$$

for a given rule set $RUL' \subseteq RUL$. The function $fitness_{\lambda,n} : P(RUL) \rightarrow \mathbb{R}^+$ is parameterized by the performance bias and the constant n , called the *cutoff-value*, which determines the range of interesting models. A plot of this function versus complexity for fixed performance is shown as the dashed line in Figure 8.1.

Using a fitness function with a cutoff-value is equivalent to only applying the genetic operators if the rule sets corresponding to the result individuals have lower complexities than the cutoff-value. Observe that this places some restrictions on the initialization of the population; most (if not all) of the individuals in the initial population should be initialized with complexities lower than the cutoff-value to avoid immediate extermination. Initialization is further discussed in Section 8.4.1. Other, more soft approaches than equation 8.7 may of course be invented, such as a strong progressive punishment of high complexity above a certain point. However, a hard cut-off value shows good performance in practice.

Reduction of the Search Space

Using a cutoff-value effectively limits the search to a small part of the search space, removing the large, uninteresting portion of the space consisting of very complex models. However, the reduction need not be dramatic, because the resulting space of small models may still be very large. Given a rule set RUL , the cardinality of the reduced search space is, for cutoff-value n :

$$\sum_{k=0}^n \binom{|RUL|}{k} = \sum_{k=0}^n \frac{|RUL|!}{k!(|RUL| - k)!} \quad (8.8)$$

As expected, equation 8.8 gives the value $2^{|RUL|}$ when $n = |RUL|$.

8.3.4 Scaling

Generally, the parents used in the genetic operations are sampled with probability proportional to fitness (see Section 8.4.4). Thus, the fitness function determines, for instance, how many times the best individual in a population will be involved in reproduction per generation, compared to the average individual. The relationship between the individuals' fitness and the average fitness in a generation is important for the future evolution of the population. Several undesirable scenarios exist. Firstly, *diversity* in the population is important to prevent premature convergence. If all individuals in the population are equal or almost equal according to the fitness function, crossover operations will fail to produce more fit individuals. Said with other words; if the average individuals are almost as fit as the fittest individual, they would reproduce nearly the same number of times and survival of the fittest is no longer maintained. Secondly, the existence of a few extremely fit (that is, with fitness much higher than the average fitness of the population) individuals is not desired in early generations. One such individual, called a *parasite*, is likely to "take over" the whole population. Since this individual is extremely likely to

reproduce and create more very fit individuals with parts of the same gene material, the other gene material in the population will rapidly be wiped out, creating a population of clones of the parasite. In effect, the consequence is the same as in the first scenario: rapid convergence to a homogeneous population. Both of these problems are due to bad *scaling* of the fitness values. In the first case, the difference between the individuals in a relatively homogeneous population should be emphasized. In early generations, the second case, the differences in fitness value should be scaled down to prevent parasites taking over the population. These problems can be helped by scaling the fitness function. The scaling scheme used here is *Boltzmann scaling*, see e.g. Mitchell (1996). Let f be the original fitness function and P the population. Then, the scaled fitness function is f' :

$$f'(I) = \frac{e^{f(I)/T}}{\frac{1}{p} \sum_{I' \in P} e^{f(I')/T}} \quad (8.9)$$

where T , called the temperature, decreases over time.

8.4 The Algorithm

A genetic algorithm for solving the rule filtering optimization problem has been implemented. Although it was argued in Section 5.2 that GAs are rather generic and that a optimization problem is solved by a GA by fitting the problem to the algorithm instead of the other way around, it is often beneficial to tailor parts of the algorithm – e.g. like the initialization or stopping criteria – to the problem at hand. This approach is shown in the function `GENETIC-OPTIMIZATION` in Figure 5.3 on page 38, where various parameters for a genetic algorithm is retrieved from – for example – a user, before solving an optimization problem. The current section describes a genetic algorithm tailored to the rule filtering problem. The implementation and the architecture presented in the following are heavily based on the architecture and implementation from Vinterbo (1999); see Appendix A.3 for implementation details.

Solving the rule filtering optimization problem using a GA is shown on an abstract level in Figure 8.3. `GENETIC-RULE-FILTERING` uses the `GENETIC-OPTIMIZATION`

```

function GENETIC-RULE-FILTERING( $RUL$ ) returns  $RUL' \subseteq RUL$ 
  inputs:
     $RUL$ , a set of rules

   $\lambda \leftarrow \text{GETPERFORMANCEBIAS}()$ 
   $n \leftarrow \text{GETCUTOFFVALUE}()$ 
   $RUL' \leftarrow \text{GENETIC-OPTIMIZATION}(P(RUL), \text{fitness}_{\lambda,n}, RUL, \iota_{P(RUL)})$ 
  return  $RUL'$ 
end function

```

Figure 8.3: Rule Filtering with a genetic algorithm

function from Figure 5.3. The inputs to the latter algorithm is the optimization problem (S, f) and the coding $(G, \text{meaning})$. As discussed above, $G = RUL$, $S = P(RUL)$ and $\text{meaning} = \iota_{P(RUL)}$. As discussed in the previous section, the evaluation function for the optimization problem is identical to the fitness function. The fitness function $\text{fitness}_{\lambda,n}$ is instantiated by the parameters collected by the functions `GETPERFORMANCEBIAS` and `GETCUTOFFVALUE`.

GENETIC-OPTIMIZATION collects a range of parameters for GENETIC-ALGORITHM (Figure 5.2 on page 37). These parameters are functions that constitutes parts of the GA. This section documents how these functions are implemented in the rule filtering genetic algorithm. The function GETGENETICPARAMETERS can be assumed to collect functions with the functionality discussed in this section. Some of these functions have associated parameters. In the implementation, the parameters, in addition to the parameters related to the fitness function described in the previous section, can be set by the user. Section 8.5 summarizes the user selectable parameters.

8.4.1 Initialization

Selecting good starting points is important in all search strategies. In genetic algorithms, the starting point is set by creating an initial population. As discussed in Section 8.3.4 the existence of super-individuals in the initial population is undesirable, but the negative effect can be remedied by scaling the fitness function.

The implemented initialization algorithm is shown in Figure 8.4. The main param-

```

function RULE-FILTER-INITIALIZATION(RUL) returns P, a population
  inputs:
    RUL, a set of rules

  size  $\leftarrow$  GETPOPSIZE()
  p  $\leftarrow$  GETINITRATE()
  for i := 1 to size do
    B =  $b_0 \cdots b_{|RUL|-1}$ 
    for j := 0 to  $|RUL| - 1$  do
      set  $b_j = 1$  with probability p
    end for
    P  $\leftarrow P \cup \{\llbracket B \rrbracket\}$ 
  end for
  return P
end function

```

Figure 8.4: GA initialization

eter controlling the initialization is the population size *size* and the *initialization rate* *p*:

$$p = \text{Prob}(\text{bit} = 1)$$

for any bit in the bitstring representing any individual in the initial population. *p* can also be seen as a normalized measure of the average complexity of an individual in the initial population:

$$p \cdot |G| = p \cdot |RUL| = \text{average complexity}$$

where *RUL* is the rule set that is to be pruned. The cardinality of an individual (the number of genes) is equal to the number of rules in the corresponding rule set, i.e. the complexity according to Equation 6.3. In Section 8.3.3, the importance of initialization with respect to the cutoff-value was mentioned. The initialization rate should place the average complexity well below the cutoff-value:

$$p \cdot |RUL| < n$$

Placing the average complexity in the middle of the interval $[0, n]$ is a good choice in practice:

$$p = \frac{n}{2 \cdot |RUL|} \quad (8.10)$$

8.4.2 Resampling

No resampling has been implemented, that is, the resampling step is equivalent to copying the entire population. The reason for this decision is explained under the recombination step below.

8.4.3 Parent Selection

The parent population used in the genetic operations is sampled from the population according to fitness, with or without replacement.

8.4.4 Genetic Operations

Two genetic operations have been implemented: n -point crossover and q -point mutation. The operations sample from the parent population with probabilities p_c and p_m , respectively, without replacement. The operators selected by the two operations are of order 2 and 1, respectively. Each operation samples tuples of the corresponding size from the parent population. For each of the sampled tuples, the operations:

- **n -point crossover:** randomly selects a n -point crossover operator (Definition 5.8 on page 36)
- **q -point mutation:** randomly selects q mutation operators (Definition 5.7 on page 36) m_{p_1}, \dots, m_{p_q} and constructs the composite genetic operator $m_{p_1} \circ m_{p_2} \circ \dots \circ m_{p_q}$.

The selected/constructed operator is applied to the tuple, and the set of all offspring from all tuples is returned.

8.4.5 Recombination

The recombination of the offspring with the original population is done by sampling – with the same algorithm as in the parent selection (Section 8.4.3) – a number of individuals corresponding to the difference between the size of the original population and the offspring set. The original population is then replaced with the union of the sampled set and the offspring set.

The combination of the above recombination scheme and no resampling of the population is similar to an algorithm described by Michalewicz (1996). See also (Vinterbo 1999).

An additional, optional recombination feature is *elitism*. When elitism is selected, the fittest individuals will always be copied to the next generation.

8.4.6 Statistics

The algorithm updates a set of statistics for each generation. The most important information is the *keep list*. The keep list is a tuple (I_1, \dots, I_{n-1}) , where n is the cutoff-value for the fitness function. An individual I_i in the keep-list is the best individual with complexity equal to i the algorithm has seen so far. In other words, after each iteration, the algorithm compares each individual in the population to the individual in the keep-list with the same complexity, and replaces the latter with the former if the former has better fitness.

Other statistics that is computed is the average fitness of the population, and the number of generations since the keep list and the average fitness was changed, respectively.

8.4.7 The Stopping Criterion

After each iteration, the algorithm checks the stopping criterion (Figure 5.2). The parameters to the implemented stopping criterion are two integers g_k and g_f . The algorithm will stop when both the number of generations since the last time the keep list changed has exceeded g_k and the number of generation since the last time the average fitness changed has exceeded g_f .

8.5 Customization

The following is a summary of the parameters related to rule filtering with the genetic algorithm:

- Fitness Function:
 - Performance bias λ
 - Cut-off value n
 - AUC calculation:
 - * Decision system \mathcal{A}
 - * ROC focus class
 - * Fallback certainty for the ROC focus class
 - Scaling:
 - * Use scaling?
 - * ΔT for each generation, for the Boltzmann function
 - * Minimum and maximum temperature for the Boltzmann function
- Genetic Operations:
 - The values n and p_c for n -point crossover with probability p_c
 - The values q and p_m for q -point mutation with probability p_m
- Initialization:
 - Initial population size
 - Initialization rate p

- Resampling:
 - Use elitism?
- Parent Selection:
 - Sample with replacement?
- Stopping:
 - Keep list generation gap g_k
 - Average fitness generation gap g_f

Part III

Results and Discussion

Predicting Acute Appendicitis

Diagnosing acute appendicitis is often difficult, even for experienced surgeons. In addition, the fact that not performing surgery on a patient *with* acute appendicitis is much more serious than performing an unnecessary operation on a patient *without* acute appendicitis helps to explain the high rate of false positive diagnoses being made. However, a high number of unnecessary operations are of course undesirable – also from an economical viewpoint. It is clear then, that better methods for classifying patients with suspected acute appendicitis are needed.

In the current chapter, a data set describing patients with suspected acute appendicitis collected at Innherred Hospital in Norway (Halland, Åsberg & Edna 1997b, Halland, Åsberg & Edna 1997a) is examined.

9.1 Data Material

Between March 1992 and January 1994, three hundred and nine patients with acute abdominal pain referred to the department of surgery by general practitioners were given acute appendicitis as one of the differential diagnoses after the initial clinical examination. Clinical data concerning each patient were recorded, and the risk of acute appendicitis estimated by a surgeon to one of ten equally spaced intervals between 0 and 1. The correct diagnoses were recorded later. Halland et al. (1997b) compare the physicians' probability estimates with a logistic regression model created from parts of the collected data material and the actual occurrences of disease. The output from the logistic regression model on previously unseen data and the physicians' probability estimates, were plotted in ROC curves. The main result from Halland et al. (1997b) is that the AUC of the two curves were not significantly different. Halland et al. (1997a) continue the examination of the data set by including data from blood samples drawn from each patient¹. In this extended data set, a total of 52 patients were excluded from the original set of 309 patients because of missing values. New logistic regression models including the new evidence were constructed. The main result from Halland et al. (1997a) is that the AUC increased significantly when biochemical tests were included in the models in addition to the clinical data.

¹The estimation of the probability of acute appendicitis was done without the results of the blood test.

Attribute	Description	Statistics	
		Yes % (count)	No % (count)
<code>sex</code>	Male sex?	55.3% (142)	44.7% (115)
<code>anorexia</code>	Anorexia?	69.3% (178)	30.7% (79)
<code>nausea</code>	Nausea or vomiting?	70.8% (182)	29.2% (75)
<code>previous</code>	Previous surgery?	9.3% (24)	90.7% (233)
<code>movement</code>	Aggravation of pain by movement?	61.5% (158)	38.5% (99)
<code>coughing</code>	Aggravation of pain by coughing?	59.9% (154)	40.1% (103)
<code>mictur</code>	Normal micturition?	87.2% (224)	12.8% (33)
<code>tendrlq</code>	Tenderness in right lower quadrant?	86.0% (221)	14.0% (36)
<code>rebtend</code>	Rebound tenderness in right lower quadrant?	55.3% (142)	44.7% (115)
<code>guard</code>	Guarding or rigidity?	30.7% (79)	69.3% (178)
<code>classic</code>	Classic migration of pain?	49.4% (127)	50.6% (130)
<code>left</code>	Shift to the left in differential count?	53.3% (137)	46.7% (120)
<code>diagnosis</code>	Final (real) diagnosis of acute appendicitis	38.1% (98)	61.9% (159)

Table 9.1: Binary Attributes

Carlin (1998) and Carlin, Komorowski & Øhrn (1998) analyzed the data set using rough set methods, and compared the results to those obtained with logistic regression and with the surgeons' estimates. The main results are twofold. First, the rough set method appeared to perform approximately equally well compared to logistic regression. Second, the results from Halland et al. (1997a) were confirmed. In addition to the first point is the fact that a rough set model is a white-box model while a logistic regression model is a black-box model. The classifier constructed by the rough set method can be investigated and interpreted to obtain new knowledge about the problem. However, the models generated by Carlin (1998) consist of hundreds of rules, making interpretation difficult. The motivation behind the current experiment is to use the rule filtering techniques presented in previous chapters on rule sets generated by methods suggested by Carlin (1998).

9.1.1 Attributes

The clinical and inflammatory parameters collected by Halland et al. (1997b) can be divided into binary attributes corresponding to yes/no questions and numerical attributes. There is a total of 19 attributes (excluding the decision attribute).

Binary Attributes

The binary attributes are listed in Table 9.1. All attributes except `left` and `diagnosis` are clinical attributes. `left` is one of the inflammatory parameters. This attribute was not taken into consideration by Halland et al. (1997b) and Halland et al. (1997a) but was considered by Carlin (1998). `diagnosis` is the correct

Attribute	Description	Unit	Discretization		
			Interval	Count	Interval Description
age	Age	years	$[-\infty, 17)$	84	Low
			$[17, 31)$	90	Middle
			$[31, \infty)$	83	High
duration	Duration of pain	hours	$[-\infty, 13)$	96	Short
			$[13, 30)$	71	Middle
			$[30, \infty)$	90	Long
temp	Rectal temperature	°C	$[-\infty, 37.5)$	87	Low
			$[37.5, 38.1)$	89	Middle
			$[38.1, \infty)$	81	High
ESR	Erythrocyte sedimentation rate	mm	$[-\infty, 10)$	126	Normal
			$[10, 25)$	99	Slightly raised
			$[25, \infty)$	32	Considerably raised
CRP	C-reactive protein concentration	mg/L	$[-\infty, 6)$	103	Normal
			$[6, 40)$	95	Slightly raised
			$[40, \infty)$	59	Considerably raised
WBC	White blood cell count	$\times 10^9$	$[-\infty, 10.0)$	91	Normal
			$[10.0, 14.0)$	80	Slightly raised
			$[14.0, \infty)$	86	Considerably raised
neutro	Neutrophil count	%	$[-\infty, 75)$	87	Normal
			$[75, 85)$	87	Slightly raised
			$[85, \infty)$	83	Considerably raised

Table 9.2: Continuous Attributes and Discretization

diagnosis; based on histological examination of the excised appendix.

Numerical Attributes

In the current experiment, the discretization of the continuous attributes proposed by Carlin (1998) is used. The numerical attributes and their discretization are shown in Table 9.2. The attributes were manually discretized; CRP, WBC and neutro by medical experts and age, duration and temp into intervals of approximately the same size by Carlin (1998).

9.1.2 Data Partitioning

The data set consisting of the 257 patients and the binary and discretized attributes constitute a decision system $\mathcal{A} = (U, C \cup \{\text{diagnosis}\})$, where U is the set of patients and the condition attributes $C = \{\text{sex}, \dots, \text{left}, \text{age}, \dots, \text{neutro}\}$ are the set of collected attributes (excluding the decision attribute). Note that \mathcal{A} denotes the discretized decision system. The decision system must be split into (at least) three smaller tables as discussed in Section 4.2. In the current experiment, the data set has been split three times and all parts of the experiment is duplicated for each of the three splits. This approach lessens the impact of randomness in the results.

The following procedure was repeated for i equal to 1, 2 and 3: The decision system \mathcal{A} was split randomly into a testing set \mathcal{A}_{V_i} and a training set \mathcal{A}_{T_i} containing ap-

\mathcal{A} (257)					
Split i	\mathcal{A}_{V_i} (85)	\mathcal{A}_{T_i} (172)			
		\mathcal{A}_{L_i} (86)		\mathcal{A}_{H_i} (86)	
38.13% (98)	1	40.00% (34)	37.21% (64)	39.54% (34)	34.88% (30)
	2	42.35% (36)	36.05% (62)	36.05% (31)	36.05% (31)
	3	41.18% (35)	36.63% (63)	36.05% (31)	37.21% (32)

Table 9.3: Partitioning of the data. The table displays percentage (and number) of patients with positive diagnosis in each table (with table cardinality).

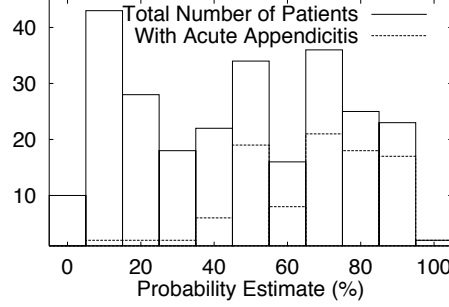


Figure 9.1: Surgeons' probability estimates.

proximately $1/3$ and $2/3$ of the objects in \mathcal{A} , respectively. The training set was then split randomly into two equally sized sets, the learning set \mathcal{A}_{L_i} and the hold-out set \mathcal{A}_{H_i} . A summary of the partitioning is shown in Table 9.3. In the table, and in the following, the term *positive diagnosis* refers to the prediction of acute appendicitis and corresponds to the value 1 of the (binary) decision attribute. As mentioned above, objects with missing values have been removed and there is thus no need for such preprocessing of the data.

9.1.3 Surgeons' Probability Estimates

In addition to the data set described above, the surgeons' estimates are available for comparison purposes. The estimates, shown in Figure 9.1, are used to construct ROC curves for analyzing the surgeons' predictions in the following.

9.2 Initial Learning

Carlin (1998) compares the performance of a range of algorithms learning decision rules within the rough sets framework, with a range of parameter settings for each algorithm. The current rule filtering experiment needs one (good) rule set to start with. To this end, the algorithm/parameters combination that seems to do best in (Carlin 1998) has been used to reproduce similar models².

²The models will of course not be identical, as we do not have access to the particular data splits used by Carlin (1998) and thus cannot reproduce 100% identical experiments.

Rule set	Rule count
$r1a_1$	423
$r1a_2$	482
$r1a_3$	436

Table 9.4: Descriptions of initial rule sets

The following is the (apparently) best settings in (Carlin 1998).

- Dynamic reduct computation, sampling 10 different samples on 5 different sets sized from 10% to 50% of the original table
- Using a genetic reduct computation algorithm to find one object related reduct modulo the decision attribute for each subtable

Using these settings, Carlin (1998) obtained an AUC value of approximately 0.91 (SE = 0.0274), averaged over 20 different testing/training splits.

Descriptions of the rule sets generated in the current experiment using the above settings for the three splits are shown in Table 9.4. The rule sets are named $r1a_i$ after the corresponding experiment by Carlin (1998), and the subscript i indicates the split number.

9.3 Motivation

The rule sets summarized in Table 9.4 are relatively small compared to typical rule sets in general rough set experiments. This fact is due to the nature of the data set and the particular algorithm/parameters used. Although the rule filtering algorithms presented in previous chapters were designed with much larger rule sets (e.g. over 100,000 rules) in mind, they are of course applicable to the current problem. Although seemingly dramatically smaller than rule sets with hundreds of thousands of rules, the sets in Table 9.4 are still large enough to demand the use of special search methods rather than brute force. The search space consisting of subsets of the smallest rule set (423 rules) has

$$2^{423} \approx 2.17 \times 10^{127}$$

states, an intractable size for conventional search techniques. The relatively small sizes of the rule sets are thus not an obstacle for illustrating the use of the rule filtering algorithms.

Some of the advantages of using the selected data set are:

- Several analyses of the data set are available, from both the medical and the computer science community
- An analysis of learning in the rough set framework is available, and analyses of the learning step can be omitted from the current experiment
- The data set has been collected from local medical expertise, potentially making interpretation of the data material and validation of the results easier

9.3.1 Goals

The main goal of the current experiment, in addition to gaining insight into the truth or falsity of the hypotheses stated in Chapter 1, is to discover descriptive models for the acute appendicitis domain. With the current definition of complexity as rule count this means that an upper limit for the size of a “descriptive” rule set must be defined.

For the current problem, we define 10 rules as the maximum size of the pruned rule sets.

9.4 Rule Quality

The quality-based rule filtering algorithm (Figure 7.1) was run with *RUL* in turn equal to each of $r1a_1$, $r1a_2$ and $r1a_3$ and *quality* equal to each of the following quality functions:

- *quality*_{Michalski} with $\mu = 0$
- *quality*_{Michalski} with $\mu = 0.5$
- *quality*_{Michalski} with $\mu = 0.8$
- *quality*_{Torgo}
- *quality*_{Brazdil}
- *quality*_{Pearson}
- *quality*_J
- *quality*_{Cohen}
- *quality*_{Coleman}
- *quality*_{C1}
- *quality*_{C2}
- *quality*_{Kononenko}

The first instance of the Michalski formula corresponds to filtering according to coverage. The G2 formula was excluded since – for sorting and filtering purposes – the J formula gives the same results.

The *performance* parameter to the filtering algorithm was set equal to $AUC_{\mathcal{A}_{H_i}}$; the AUC evaluated on the hold-out set corresponding to the rule set. The classifier scheme outlined in Section 3.6 was used to classify the hold-out set. In the case of an empty set of firing rules for an object, $\phi(x)$ (the certainty of a positive diagnosis) was set to the a priori probability of a positive decision; the positive decision rate from the learning set (see Table 9.3).

Below, some properties related to the computed quality measures are presented. The actual rule filtering is discussed in Section 9.6.

9.4.1 Quality Distributions

The algorithm in Figure 7.1 generates a graph for a finite set of quality values smaller than or equal to the size of the unfiltered rule set. Generally, several rules can be assigned identical quality values. The distributions for all the quality formulas for the three rule sets are shown graphically in Appendix B.1. Table 9.5 contains a summary of the distributions.

The only quality formula that gave undefined measures for any of the three rule sets was the J-measure. The relatively large number of rules with undefined quality (over fifty) renders the first approach to dealing with undefined quality presented in Section 7.3.2 inadequate for the current rule filtering purposes. Thus, for the J-measure, the other approach – removing rules with undefined quality from the initial rule set – must be used.

The average number of rules per quality value determines the *resolution* of the filtering algorithm. The average number over the best values is especially important for finding small rule sets. It seems like the number of rules for the first ten quality values are acceptable except for the two formulas with high best value counts; Coleman and Kononenko. The Coleman formula gives over fifty rules the same, best, quality value, rendering it useless for finding rule sets smaller than that size. The same is true for the Kononenko formula, regarding rule sets smaller than eleven to eighteen rules. There is no guarantee that, given only the best counts, the other formulas behave well for rule filtering purposes, but the low average counts for the ten best rules support this conjecture.

9.5 Genetic Computations

The genetic rule filtering algorithm in Figure 8.3 was run with RUL in turn equal to $r1a_1$, $r1a_2$ and $r1a_3$. In the implementation of the algorithm (see Appendix A.3), statistics for each generation are written to a log file. In the current section, the many parameter settings for the algorithm (see Chapter 8) are discussed, followed by a presentation of some of the experimental results related only to the genetic algorithm. The main results in the rule filtering context are presented in Section 9.6.

9.5.1 Parameter Settings

Fitness Function and Initialization

The complexity of a filtered model RUL' is $complexity_{RUL}(RUL')$, where $RUL = r1a_i$ is the unfiltered rule set (the input to the algorithm in Figure 8.3). $performance(RUL')$ is equal to $AUC_{A_{H_i}}(RUL')$; the AUC computed relative to the hold-out set corresponding to $r1a_i$ computed using the classifier scheme outlined in Section 3.6. The fallback certainty for the positive diagnosis was set equal to the a priori probability from the corresponding learning set.

Carlin (1998) obtains AUC values in the neighborhood of 0.9 with the same rule generation algorithms used in here. Assuming that the unfiltered rule sets generated in the current experiment exhibit similar performance on the testing sets³, the

³This can of course not be investigated at this point of the experiment without statistically violating the experiment.

Quality measure	Rule set	Undefined count	Best count	Average count	
				All val.	10 best val.
Michalski ($\mu = 0$)	$r1a_1$	0	2	7.83	1.70
	$r1a_2$	0	1	8.96	1.50
	$r1a_3$	0	1	7.93	1.50
Michalski ($\mu = 0.5$)	$r1a_1$	0	1	2.19	1.00
	$r1a_2$	0	1	2.49	1.00
	$r1a_3$	0	1	2.07	1.00
Michalski ($\mu = 0.8$)	$r1a_1$	0	1	2.14	1.50
	$r1a_2$	0	1	2.45	1.60
	$r1a_3$	0	1	2.04	1.40
Torgo	$r1a_1$	0	1	2.08	1.40
	$r1a_2$	0	1	2.40	1.60
	$r1a_3$	0	1	2.02	1.40
Brazdil	$r1a_1$	0	1	2.08	1.10
	$r1a_2$	0	1	2.40	1.00
	$r1a_3$	0	1	2.02	1.10
Pearson	$r1a_1$	0	1	2.11	1.00
	$r1a_2$	0	1	2.41	1.10
	$r1a_3$	0	1	2.06	1.20
J	$r1a_1$	51	1	2.11	1.00
	$r1a_2$	63	1	2.42	1.20
	$r1a_3$	63	1	2.01	1.40
Cohen	$r1a_1$	0	1	2.20	1.00
	$r1a_2$	0	1	2.42	1.20
	$r1a_3$	0	1	2.13	1.40
Coleman	$r1a_1$	0	51	4.11	6.00
	$r1a_2$	0	63	3.98	7.50
	$r1a_3$	0	63	3.55	7.40
C1	$r1a_1$	0	1	2.08	1.40
	$r1a_2$	0	1	2.40	2.30
	$r1a_3$	0	1	2.02	1.70
C2	$r1a_1$	0	1	2.08	1.40
	$r1a_2$	0	1	2.40	1.60
	$r1a_3$	0	1	2.02	1.40
Kononenko	$r1a_1$	0	11	3.36	2.40
	$r1a_2$	0	18	3.95	4.10
	$r1a_3$	0	16	3.55	4.10

Table 9.5: Breakdown of the quality value distribution. The table displays the number of rules with undefined quality, the number of rules with the (same) best quality (leaving out the rules with undefined quality), the average number of rules over all the quality values and the average number of rules over the ten best quality values (again leaving out the rules with undefined quality).

Rule set	Performance bias	Initialization Rate
$r1a_1$	0.2985	0.0236
$r1a_2$	0.2719	0.0207
$r1a_3$	0.2922	0.0229

Table 9.6: Performance biases and Initialization Rates

goal is to find small enough models with little or no drop in performance. This can be used as a starting point for setting the performance bias.

We choose to define that a ten rule model with an estimated AUC equal to 0.90 and a one rule model with an estimated AUC equal to 0.85 are equally desirable for our purposes. That is, if we have a pruned model with 10 rules (stated as the maximum size for the interesting models) and AUC equal to 0.90, a one rule model must at least have an AUC equal to 0.85 to be chosen instead. Given a rule set RUL , using equation 8.6:

$$\begin{aligned}\Delta c &= 1 - \frac{1}{|RUL|} - (1 - \frac{10}{|RUL|}) = \frac{9}{|RUL|} \\ \Delta p &= 0.85 - 0.90 = -0.05 \\ \lambda &= \frac{\frac{9}{|RUL|}}{\frac{9}{|RUL|} + 0.05}\end{aligned}$$

or

$$\lambda = \frac{9}{9 + 0.05|RUL|} \quad (9.1)$$

The values for the performance bias for the three rule sets, given by equation 9.1, are shown in Table 9.6.

The cutoff value was selected equal to 20; higher than the maximum rule set size both to increase the diversity of the population and in the case that there exist an extremely good rule set just above the ten rules limit.

The initialization rate p for each split are calculated from equation 8.10, and shown in Table 9.6.

General Parameters

Table 9.7 summarizes the general parameter settings (see Section 8.5).

Increasing the population size will never decrease the performance of a genetic algorithm. The size equal to 400 was chosen as the largest size it was practical to use on the available computer hardware. See Appendix A.3 for comments regarding memory usage. The algorithm was set to terminate if no changes were detected for five hundred generations.

The rest of the parameters have been given default values that have shown to be reasonable with other data sets.

Parameter	Value
Use scaling?	Yes
ΔT	0.02
Minimum T	1.45
Maximum T	6.45
n -point crossover	1
Crossover probability p_c	0.3
n -point mutation	1
Mutation probability p_c	0.05
Initial population size	400
Use elitism?	Yes
Sample with replacement?	Yes
Keep list generation gap g_k	500
Average fitness generation gap g_f	500

Table 9.7: Parameter settings

Rule set	AUC	Generations	Best fitness		
			Value	Cardinality	AUC
$r1a_1$	0.8833	2475	0.9758	8	0.9634
$r1a_2$	0.9507	3281	0.9833	9	0.9880
$r1a_3$	0.8935	4501	0.9733	9	0.9586

Table 9.8: Genetic algorithm statistics, showing the AUC on the hold-out sets for the unfiltered rule sets and for the rule sets with best fitness after termination.

9.5.2 GA Performance

Here, results illustrating the genetic algorithm's ability to solve an optimization problem is presented; generally as properties varying with the generation number.

Table 9.8 displays key statistics for the genetic algorithm runs. The table shows the number of generations used to converge (including the generation gap used by the stopping criterion), the fitness of the best individual found (the general solution) and the number of rules and the AUC for the latter. The AUC values show that the algorithm fits the pruned rule sets very closely to the hold-out set, indicating possible overfitting. The required number of generations is large, in addition to that the evaluation – the AUC calculation to be more precise – is very computational intensive. Each of the runs therefore took several hours to complete.

Figure 9.2 displays the fitness of the most fit individual so far as a function of the number of generations, for each split. The figure also shows breakdowns of the fitness curves into the constituencies of the fitness function. In Figure 9.3, the rate of change in the keep list and the population is depicted. The curves give, for each generation, the number of past generations where changes were detected in the keep list or in the average fitness.

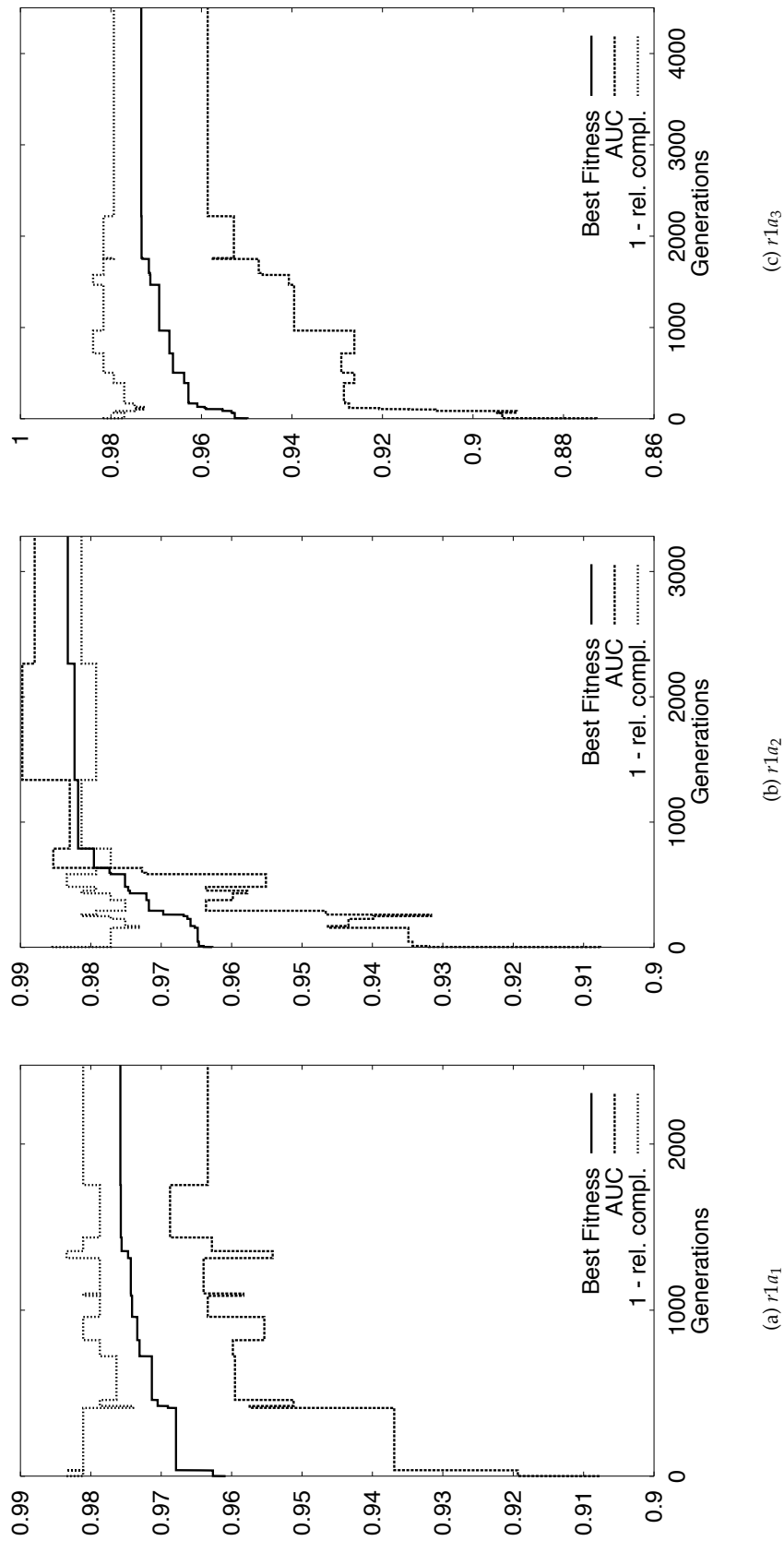


Figure 9.2: Performance, complexity and fitness

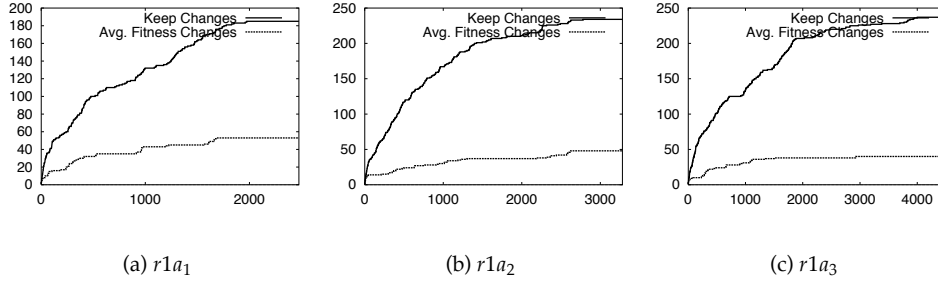


Figure 9.3: The number of generations that the keep list or the average fitness changed.

9.5.3 The Search Space

The cutoff value was set to 20 for practical reasons (Section 9.5.1), but even for cut-off equal to ten rules (the subjectively selected maximum rule set size), the search space is still large. Equation 8.8 gives the size of the (reduced) search space for the smallest rule set ($r1a_1$):

$$\sum_{k=0}^{10} \frac{423!}{k!(423-k)!} \approx 4.65 \times 10^{19}$$

The genetic algorithm, evaluating 2475 generations each of 400 individuals, explored at most⁴

$$2475 \times 400 = 9.9 \times 10^5$$

of these states. The theoretically highest percentage of the search space possibly explored by the genetic algorithm is thus

$$\frac{9.9 \times 10^5}{4.65 \times 10^{19}} \approx 2.13 \times 10^{-12}\%$$

In other words, even with a relatively small rule set and a small cut-off value, using a genetic search algorithm is justified because the search space still is too large for conventional search algorithms. This is illustrated by the fact that the genetic algorithm, iteratively evaluating states for several hours, only was able to cover a very small fraction of the search space. In the next section, whether this fraction included the important parts of the search space is investigated.

9.6 Rule Filtering

In this section, the rule filtering results are presented and discussed. In section 9.6.1 models selected with an *ad-hoc* procedure are evaluated and compared, and in section 9.6.2 further comparisons of the relative performance of the different algorithm variations are made.

⁴The actual (real) number is considerably lower, because some individuals appears in more than one generation (e.g. as a consequence of elitism).

9.6.1 Model Selection

Both rule filtering methods discussed herein can generate a range of models of varying complexity. In practical applications, a method for selecting a particular model is needed. In the current section, an *ad-hoc* model selection method is used.

The following domain specific questions are considered:

- can we find computer models based on clinical and biochemical attributes predicting acute appendicitis as well as (or better than) surgeons using clinical evidence only?
- can we find very small, interpretable models predicting acute appendicitis as well as, or better than, surgeons?

In addition, the following method specific questions are considered. Using the *ad-hoc* model selection method,

- do the small filtered models perform as well as the unfiltered models?
- which (if any) of the quality formulas performs best?
- can we find better small models with the genetic algorithm than with the best quality formula?

Quality Filtering

The graphs computed by the quality filtering algorithm are displayed in Figure 9.4, showing the relationship between rule set size and AUC computed on the hold-out sets for the various quality formulae. In the graphs for the J-measure, the rules with undefined quality were removed before filtering.

For each graph in the figure, an as small as possible model – no larger than the stated maximum rule set size of interest (see Section 9.3.1) – without significantly lower performance than the unfiltered rule set, was attempted to be found. The selected models are represented by vertical dotted lines in Figure 9.4, and they were selected as follows.

First, to be useful for rule filtering a graph must have a certain shape, as briefly discussed in section 7.3.1. Second, even a perfectly smooth graph is unusable if the whole section representing the small rule sets of interest has a significantly lower performance estimate compared to the unfiltered rule set, that is, if the graphs fall too much before the interesting section. These two properties were used to subjectively categorize the graphs into the following five categories:

1. Unusable; not shaped like a non-increasing function
2. Almost unusable
3. Good overall shape, but too much ripple in the interesting section
4. Good shape, but falls too early
5. Good

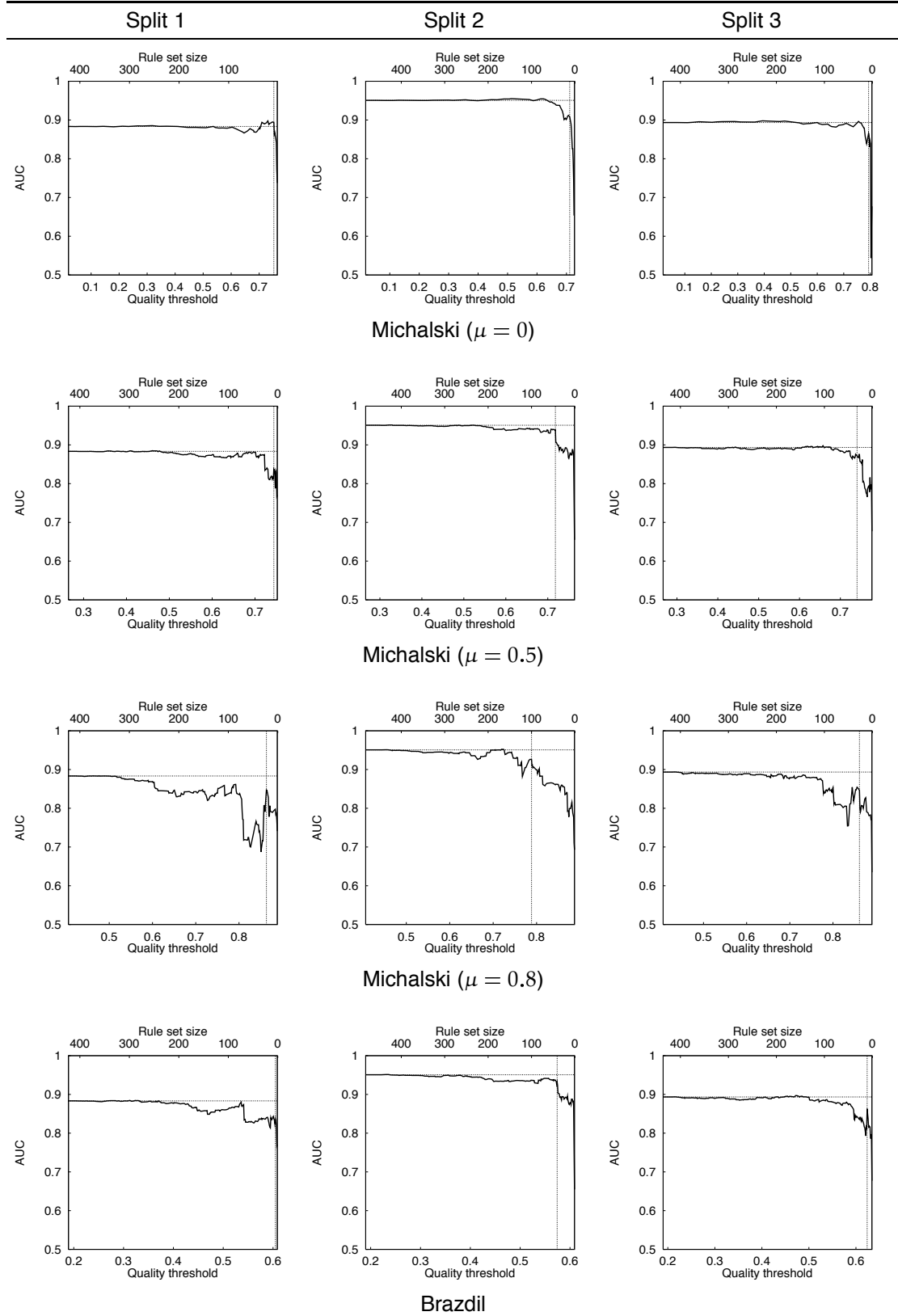


Figure 9.4: Quality Filtering. The horizontal dotted line represents the AUC for the unfiltered rule set. (Continues on page 85)

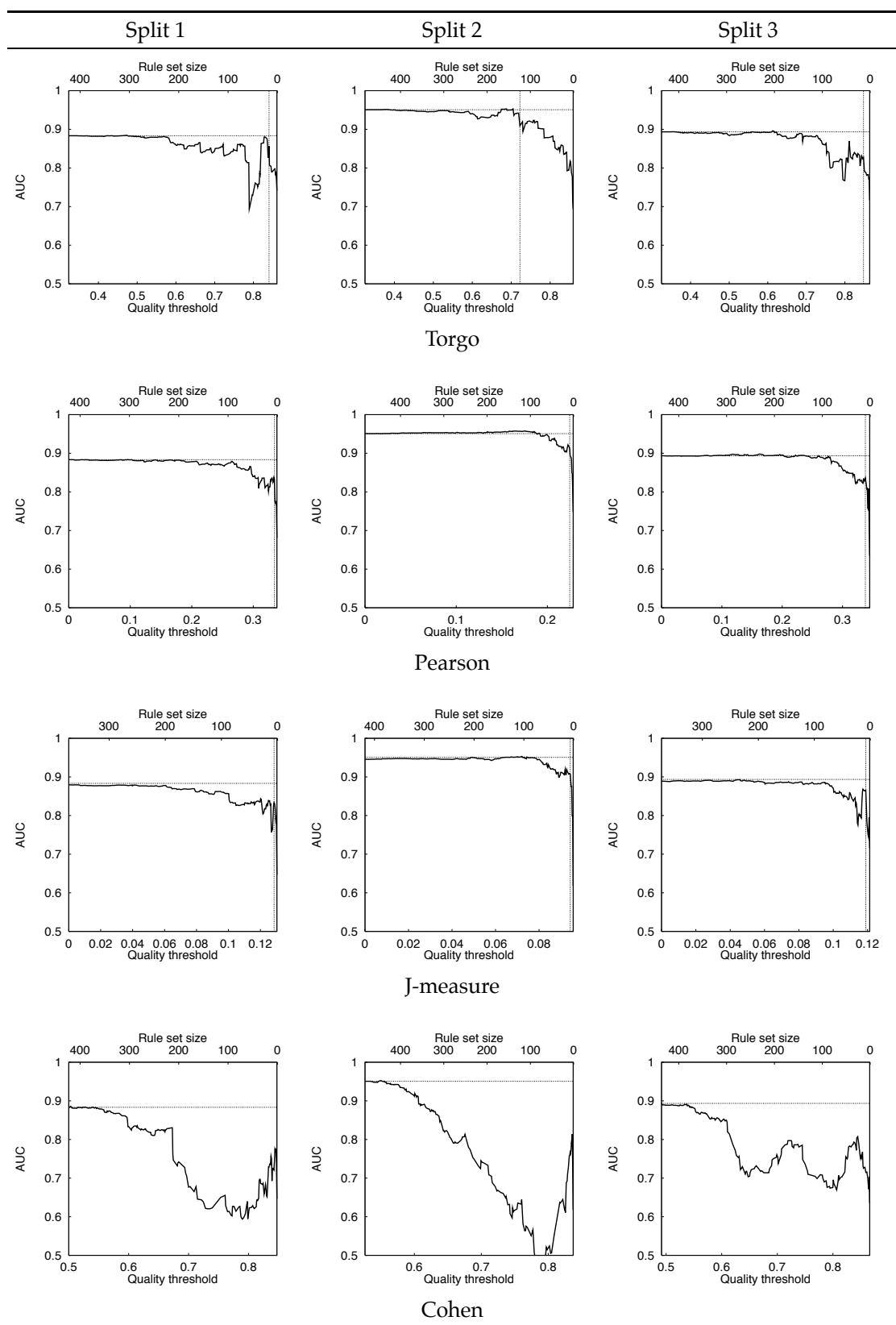


Figure 9.4: (Continued, continues on page 86)

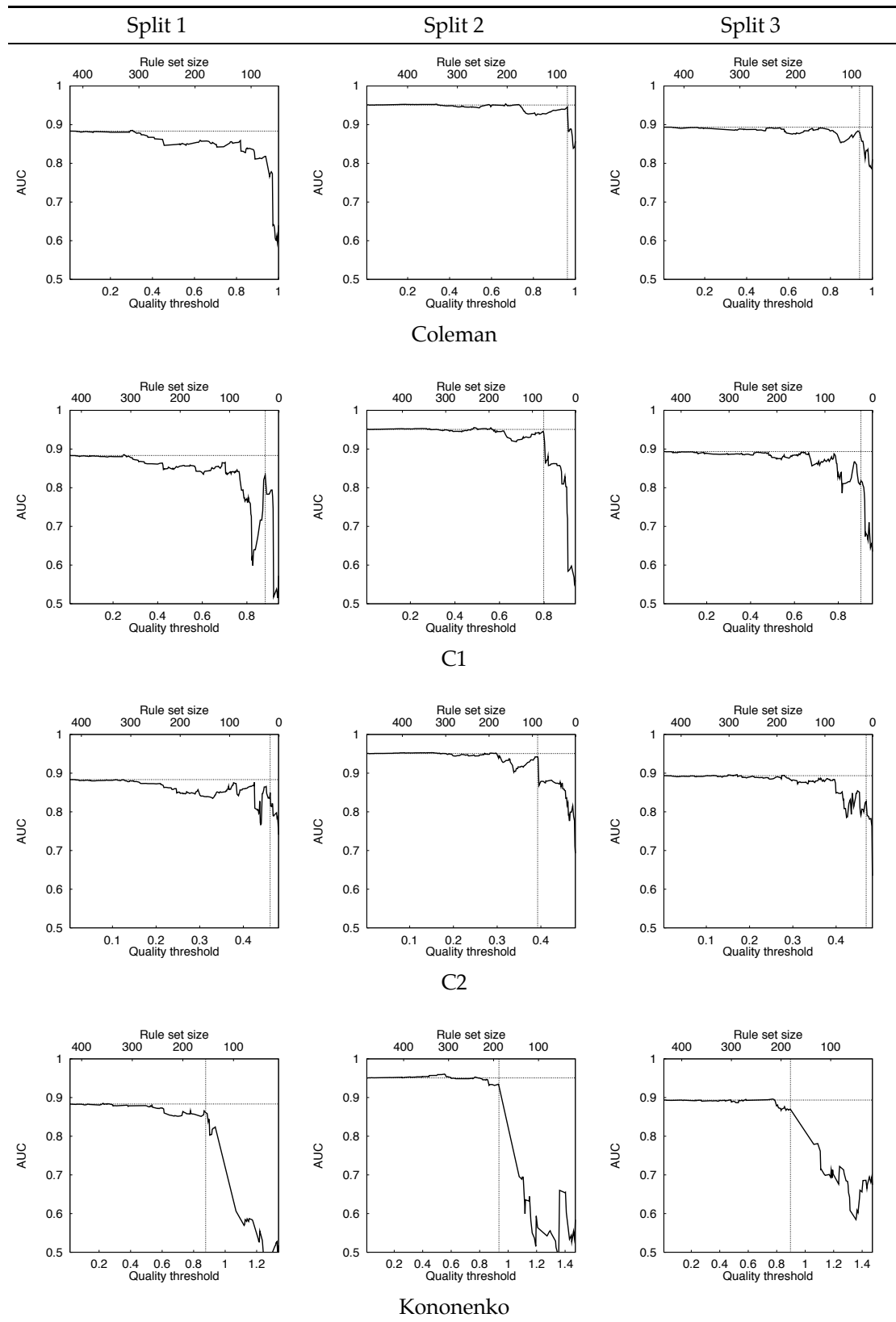


Figure 9.4: (Continued)

The first category consists of the graphs subjectively selected as being unusable for quality filtering because of unsuitable shapes. The three graphs for the Cohen formula were the only ones selected for this category. The second category consists of the graphs that were on the borderline of being put into the first category; including Michalski with $\mu = 0.8$ for splits 1 and 3, Torgo for splits 1 and 3, C1 for splits 1 and 3 and C2 for split 1. These graphs have overall tendencies to monotonically non-increasing shapes. The third category is the graphs displaying an acceptable overall monotony, but having too much ripple in the section of the graph representing the smallest rule sets (the rightmost section of the graphs as displayed in Figure 9.4). These graphs included the J-measure for splits 1 and 3, C2 for split 3 and Kononenko for splits 2 and 3. The graphs that were evaluated to be generally non-increasing but falling too early, the fourth category, were Michalski with $\mu = 0.5$ for splits 2 and 3, Michalski with $\mu = 0.8$ for split 2, Brazdil for split 2, Torgo for split 2, Coleman for all splits (because of high best count; see Table 9.5), C1 for split 2, C2 for split 2, and Kononenko for split 1. The fifth and last category is the “good” graphs, the relatively monotonically non-increasing graphs with knee-points in the interesting region.

No models were selected for the graphs in the first category. The Cohen formula was thus deemed unusable for the current experiment. The formulas in the second and third categories are of questionable value given the evidence in the graphs. Nevertheless, models were selected for these graphs, by selecting either a knee-point or a general peak exhibiting no significant drop in performance. For the graphs in the fourth category, a knee-point or, more generally, a point immediately before significant performance drop, was selected. In categories two, three and four, a model with insignificant performance drop was not always available in the section of the graph consisting of the very small models. In these cases, a (larger) model was nevertheless selected, for comparison purposes. These larger models do not fulfill the requirements stated for interesting models; they do, however, generally represent a relative large decrease in complexity compared to the full models. For the fifth category, the smallest models without significant performance drop were generally selected.

Significant differences in performance were tested using Hanley / McNeil’s hypothesis test (Section 4.5.2), generally with a significance level greater than 5%. The p-values for the different models were taken into consideration when selecting a model; e.g. if a model with one more rule than another model had much higher p-value, the larger model was often chosen. The p-values are one-tailed and were computed using the Pearson correlation measure.

The chosen models are summarized in Table 9.9.

In Table 9.9, and henceforth, a p-value corresponds to the result from the hypothesis test outlined in Section 4.5.2 with AUC_1 equal to the AUC of the unfiltered set and AUC_2 equal to the AUC of the filtered set, except when the AUC of the filtered set in fact is *higher* than the AUC of the unfiltered set. In this latter case, the test is turned around and AUC_1 is the AUC of the filtered set. p-values for this test are marked with a “*”.

Genetic Filtering

Graphs similar to those computed by the quality filtering algorithm were plotted from the keep-lists at the termination of the genetic algorithm, and are shown in Figure 9.5.

Formula	Split	Threshold	Size	AUC (SE)	p-value
Unfiltered	1		423	0.8833 (0.0428)	
Unfiltered	2		482	0.9507 (0.0281)	
Unfiltered	3		436	0.8935 (0.0400)	
Michalski ($\mu = 0$)	1	0.6346	9	0.8938 (0.0410)	0.4004*
	2	0.5484	12	0.9082 (0.0378)	0.0689
	3	0.6182	8	0.8655 (0.0444)	0.2120
Michalski ($\mu = 0.5$)	1	0.7115	8	0.8375 (0.0494)	0.0859
	2	0.6358	45	0.9390 (0.0312)	0.3058
	3	0.6784	21	0.8620 (0.0449)	0.1453
Michalski ($\mu = 0.8$)	1	0.8388	23	0.8473 (0.0481)	0.1798
	2	0.7713	100	0.9261 (0.0341)	0.1820
	3	0.8430	27	0.8466 (0.0470)	0.1008
Brazdil	1	0.5725	5	0.8360 (0.0423)	0.1121
	2	0.4773	41	0.9317 (0.0329)	0.2640
	3	0.5464	11	0.8631 (0.0447)	0.2176
Torgo	1	0.8106	17	0.8566 (0.0469)	0.2460
	2	0.6975	124	0.9091 (0.0376)	0.0866
	3	0.8239	14	0.8273 (0.0494)	0.0572
Pearson	1	0.2772	6	0.8336 (0.0499)	0.0770
	2	0.1938	9	0.9114 (0.0372)	0.1176
	3	0.2292	10	0.8359 (0.0484)	0.0916
J	1	0.1049	6	0.8336 (0.0499)	0.0770
	2	0.0831	7	0.9073 (0.0380)	0.0865
	3	0.0877	8	0.8660 (0.0443)	0.2221
Coleman	1	0.6578	119	0.8589 (0.0465)	n/a
	2	0.8045	79	0.9455 (0.0295)	0.4240
	3	0.7766	86	0.8808 (0.0421)	0.3065
C1	1	0.8746	28	0.8327 (0.0500)	0.1192
	2	0.7518	74	0.9411 (0.0306)	0.3670
	3	0.8733	25	0.8183 (0.0505)	0.0544
C2	1	0.4142	18	0.8494 (0.0478)	0.1582
	2	0.3260	88	0.9422 (0.0303)	0.3783
	3	0.4257	14	0.8273 (0.0494)	0.0484
Kononenko	1	0.6384	155	0.8610 (0.0462)	0.2161
	2	0.6449	188	0.9299 (0.0333)	0.1899
	3	0.6449	181	0.8695 (0.0438)	0.2160

Table 9.9: Selected models for quality based rule filtering. The AUC values are computed relative to the hold-out sets. p-values are given relative to the unfiltered rule sets, and values marked with a “*” indicate that the AUC for the filtered set is higher than the AUC for the unfiltered set. The p-values are one-tailed, computed using Pearson correlation. For fields marked “n/a”, the p-value could not be computed because the corresponding look-up values for the Hanley/McNeil correlation table were outside the scope of the table.

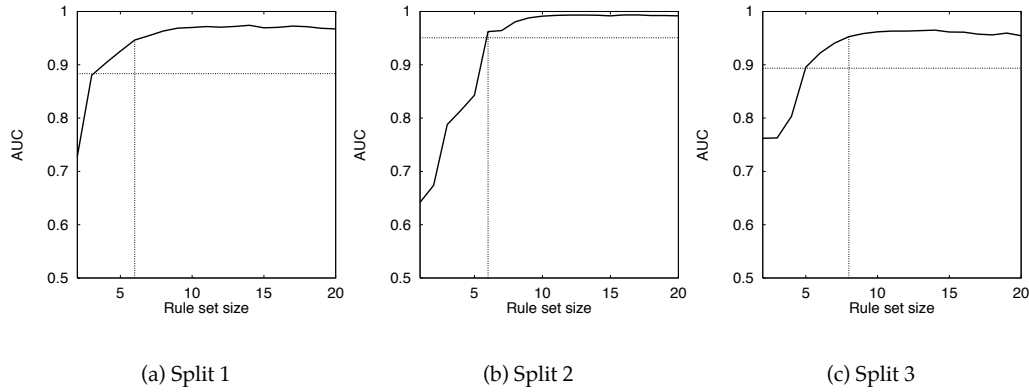


Figure 9.5: Genetic filtering. The horizontal dotted line represents the AUCs for the unfiltered rule sets.

Similarly to the quality based filtering, a model was chosen for each of the graphs for the genetic algorithm; represented by vertical dotted lines in Figure 9.5. The models were subjectively selected at the knee-points of the curves, and are summarized in Table 9.10⁵.

Split	Size	AUC (SE)
1	6	0.9464 (0.0297)
2	6	0.9621 (0.0247)
3	8	0.9528 (0.0271)

Table 9.10: Selected models for genetic rule filtering

Evaluation

The selected models from the quality and genetic based algorithms were evaluated on the testing set for each split, and statistically compared to both the unfiltered rule sets and the surgeons' probability estimates. Henceforth, statistical significance will be considered at the 5% significance level for the one-tailed test. Only selected classifiers containing no more than twenty rules were evaluated, since only small classifiers are of interest. The results are shown in Table 9.11.

Some general comments regarding the suitability of the quality formulas follows. The selected classifier for the only split where Michalski with $\mu = 0.5$ does not fall off too early has poor performance. In addition to giving relatively large models, Michalski with $\mu = 0.8$ does not generate well-behaved graphs. For the two splits where Brazdil, Torgo and C2 do not fall off too early, the selected classifiers exhibit poor performance (all) or the graphs are in addition not well-behaved (Torgo and C2). Coleman, C1 and Kononenko give too large models, and the two latter

⁵Note that the AUC values estimated on the hold-out sets are higher for the genetic algorithm compared to the quality based algorithm, because the genetic algorithm used the hold-out sets to select models with "good" features.

Method	Split	AUC (SE)	p-value	
			Unfiltered	Surgeons
Unfiltered	1	0.8726 (0.0424)		0.4480*
	2	0.9085 (0.0355)		0.0396*
	3	0.9317 (0.0312)		0.0087*
Surgeons	1	0.8653 (0.0435)	0.4480	
	2	0.8194 (0.0484)	0.0396	
	3	0.8154 (0.0493)	0.0087	
Genetic	1	0.8815 (0.0410)	0.4154*	0.3880*
	2	0.9141 (0.9141)	0.4384*	0.0485*
	3	0.8834 (0.0403)	0.0775	0.1156*
Michalski ($\mu = 0$)	1	0.8671 (0.0432)	0.4351	0.4872
	2	0.9133 (0.0346)	0.4348*	0.0263
	3	0.8866 (0.0398)	0.0593	0.0969
Michalski ($\mu = 0.5$)	1	0.8091 (0.0505)	0.0406	0.1880
Brazdil	1	0.8022 (0.8022)	0.0207	0.1627
	3	0.8829 (0.0403)	0.0847	0.1277*
Torgo	1	0.8365 (0.0473)	0.1905	n/a
	3	0.8629 (0.0434)	0.0363	0.2178*
Pearson	1	0.8333 (0.0477)	0.1546	0.3066
	2	0.8708 (0.0418)	0.1767	0.2008*
	3	0.8943 (0.0385)	0.1567	0.0935*
J	1	0.8333 (0.0477)	0.1546	0.3066
	2	0.8597 (0.0433)	0.1291	0.2604*
	3	0.9271 (0.0322)	0.4383	0.0171*
C2	1	0.8639 (0.0437)	0.4087	n/a
	3	0.8629 (0.0434)	0.0363	0.2178*

Table 9.11: The small selected models' performance on the testing sets with statistical comparisons. The AUC values are computed relative to the testing sets. p-values are given relative to the unfiltered rule sets, and values marked with a "*" indicate that the AUC for the filtered set is higher than the AUC for the unfiltered set. The p-values are one-tailed, computed using Pearson correlation. For fields marked "n/a", the p-value could not be computed because the corresponding look-up values for the Hanley/McNeil correlation table were outside the scope of the table.

formulas are in addition not very well-behaved. The J-measure exhibits good performance, even with the selection of peak-values because of ripple in performance for the lower rule set sizes. An additional observation is that the selected rule sets for the J-measure and the Pearson formula for split 1 (6 rules) are identical. The graphs are also generally very similar.

The statistical analysis given in Table 9.11 attempts to answer the questions given at the start of the current section (p. 9.6.1):

- Yes, the unfiltered models generated from the available clinical and biochemical tests perform as well as surgeons using only the clinical data. No, it can not be concluded that the computer models perform better than surgeons. The estimated AUC values are better, but the difference is not significant for all splits.
- Yes, we can find small models performing as well as surgeons. For example, the models generated by the genetic algorithm have 6–8 rules and all perform at least as well as the surgeons. No, it can not be concluded that the small models perform better than the surgeons. The selected models for both the genetic algorithm and quality filtering with Michalski ($\mu = 0$) exhibit higher estimated AUC values, but the difference is not significant for all splits.
- Yes, very small models performing as well as the large models can be found. The models selected for the genetic algorithm and the quality based algorithm with formulas Michalski ($\mu = 0$), Pearson and the J-measure all have at worst insignificantly poorer performance estimates compared to the unfiltered models. These four schemes are the only ones that gave small models for all splits using the *ad-hoc* model selection method described above.
- Michalski with $\mu = 0$, Pearson and the J-measure are the best quality formulas when the *ad-hoc* model selection method is used. None of these formulas perform significantly better than the others, but Michalski has both higher average estimated AUC value (0.8890 against 0.8661 (Pearson) and 0.8734 (J)) and average p-value (0.3097 against 0.1627 (Pearson) and 0.2407 (J)).
- No, using the *ad-hoc* selection method, the models selected for the genetic algorithm were not significantly better than those selected for the best quality formulas. The genetic models were in fact both smaller and better for all the splits, but the difference is not significant⁶.

Figure 9.6 shows the ROC curves for the classifiers selected for the four algorithm variants genetic, Michalski ($\mu = 0$), Pearson and J, on the testing sets.

As discussed in Chapter 4, ROC analysis compares discriminatory ability for all possible thresholds τ for the classifiers' threshold functions (see equation 3.9). For practical classification purposes, a particular threshold value must be selected, dependent upon the relative costs of the two types of error. Table 9.12 displays properties of the various classifiers when τ is chosen corresponding to the point (1, 1) in the ROC curve, measured on the testing sets. See Chapter 4 for definitions of the properties.

⁶p-values for genetic versus Michalski with $\mu = 0$: 0.3685, 0.4918 and 0.4705, for splits 1, 2 and 3, respectively.

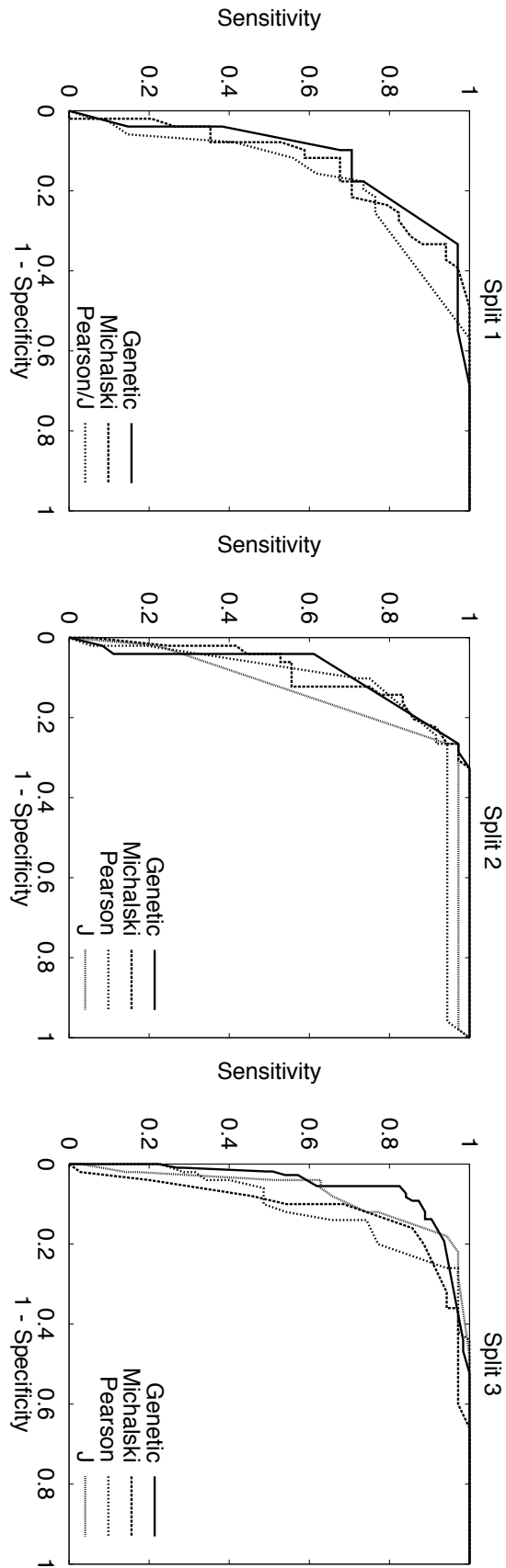


Figure 9.6: ROC curves for the selected classifiers on the testing sets. The selected classifiers for Pearson and J in split 1 are identical.

Method	Split	τ	Sensitivity	Specificity	Accuracy
Genetic	1	0.364	0.8235	0.7647	0.7882
	2	0.352	0.9722	0.7347	0.8353
	3	0.468	0.8730	0.9083	0.8954
Michalski ($\mu = 0$)	1	0.452	0.8235	0.7451	0.7765
	2	0.368	0.8333	0.8571	0.8471
	3	0.412	0.8571	0.8400	0.8471
Pearson	1	0.776	0.7353	0.8235	0.7882
	2	0.356	0.9167	0.7551	0.8235
	3	0.540	0.9714	0.7400	0.8353
J	1	0.776	0.7353	0.8235	0.7882
	2	0.224	0.9722	0.7347	0.8353
	3	0.364	0.9429	0.8200	0.8706

Table 9.12: Classifier properties for τ corresponding to the points closest to $(0, 1)$ on the ROC curves.

9.6.2 Further Comparisons

The results from the previous section were dependent upon the method used for selecting a proper submodel. In other words, the conclusions said something about the combination of the algorithm variations and the *ad-hoc* selection procedure. These conclusions are important, because a concrete selection method must be used in any rule filtering application. However, it is also interesting to say something about the different algorithm variations, *independent* of the selection method, mainly because such conclusions would be valid for other selection methods that might be employed in future applications. For example, in a particular application, the goal might be finding models with very low complexity (say, three rules) performing only approximatively as well as the unfiltered models. The goal when comparing the algorithm variations alone is thus answering the following question:

- Do any of the methods perform better than the others, independent of the (small) selected model size?

To compare the relative performance of the different algorithm variations over a range of submodel sizes without using a specific selection method, all submodels with no more than ten rules for each of the algorithm variations were used to classify the testing sets. For the quality based algorithm, this procedure is essentially the same as the one used for comparing performance on the hold-out sets in Section 9.6.1, because neither the hold-out sets nor the testing sets have been “seen” by the quality based algorithm before the model selection. However, in Section 9.6.1 it was focused on the general trends generated by the quality algorithm (Figure 9.4) and not on comparing performance over the range of small models only. Also, the similar analysis performed for the genetic algorithm in Section 9.6.1 is not comparable because the genetic algorithm has “seen” the hold-out sets during the filtering phase. To be able to compare performance on unseen data, the testing sets were used.

The results are shown plotted in Figure 9.7. It is extremely difficult to draw any general, statistically valid conclusions from the results. The following is a brief discussion of qualitative trends subjectively interpreted from inspection of the test

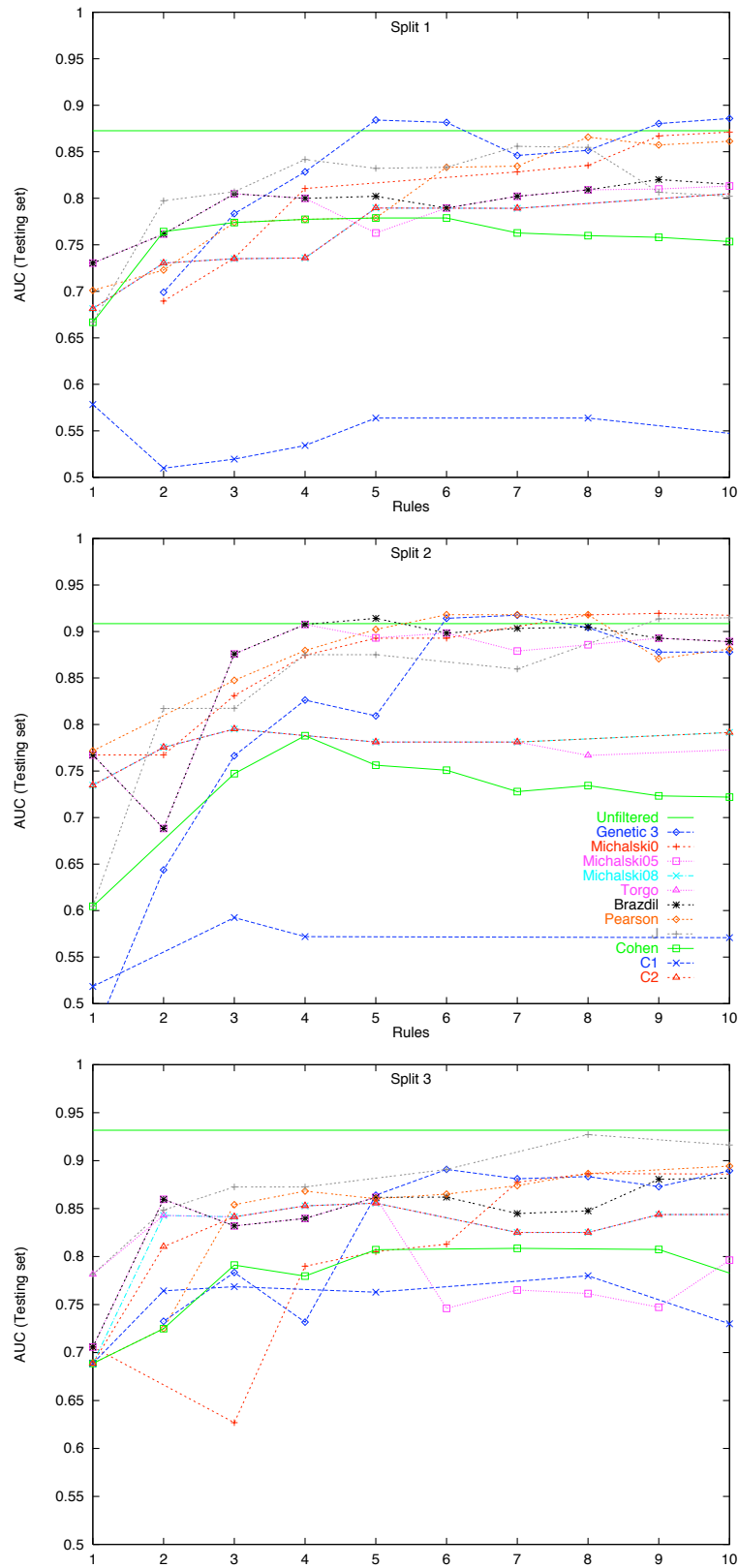


Figure 9.7: Comparisons of performance on the testing sets (see split 2 for legend)

results.

The only answer provided for the above question is:

- *No*, none of the methods stand out as better than the others for all small rule sets.

Of the three quality based variations identified as good candidates in the previous section, Michalski with $\mu = 0$ and the J-measure generally shows relatively good, but uneven, performance, while Pearson seems to show constant good performance for all three splits. Brazdil also shows tendencies to good performance, but is also uneven. The genetic algorithm generally seems to perform very well on the largest sets, but equally bad on the smallest sets. On the other side of the scale, C1 stands out as the least suitable quality formula, followed by Cohen with generally low performance. C2 and Michalski with $\mu = 0.8$ also show low performance. Coleman and Kononenko did not produce models with only ten rules or less (due to high best counts).

9.7 Discussion

In the experiment design used in this chapter, the selection procedure used with the genetic algorithm is not directly comparable to the one used with the quality based algorithm. The submodels for the latter algorithm are selected by estimating performance on unseen data, and those for the former on already seen data (hence the relatively smooth graphs in Figure 9.5 compared to Figure 9.4). It is, of course, possible to use other experiment designs and selecting models for the genetic algorithm from unseen data; e.g. by further splitting the hold-out set. However, the experience from the selection of quality based models indicates that it is difficult to construct a objective selection criterion when models are evaluated on unseen data, due to ripple in the graphs. Using the same data set used for estimating performance in the filtering phase enables the construction of a more objective selection criterion.

A statistical objection to the experiment design is that *all* the data has, in principle, been seen before the testing phase, violating the principle of testing on unseen data. The parameters for the rule learning step were collected as the best parameters from Carlin (1998), but Carlin used the whole data set to first train and then test the resulting rule sets to compare the relative performance. So, some of the data used as a part of the training phase for setting the training parameters (in Carlin's experiment) can potentially have been used to estimate the performance of the results (of the current experiment). This would have been a genuine problem in a study of the *absolute* performance measures. However, in the current *comparative* study it is very unlikely that this fact has any influence on the conclusions regarding the *relative* performance of the different algorithm variations.

Carlin (1998) reports that the introduction of biochemical tests improved the computer models, but presents no statistical analysis to support this. The results from the current study agrees with the previous studies (Carlin et al. 1998, Halland et al. 1997b) in that computer models perform at least as well as medical doctors. In addition, the computer models (based on both clinical and biochemical data) performed significantly better than doctors in two out of three cases.

CHAPTER 10

Predicting Coronary Artery Disease

10.1 Introduction

In this chapter, methods from rough set theory are used to induce models from a data set describing patients with suspected coronary artery disease. The goal is to prune these models as much as possible while retaining classifier performance. No upper bound on acceptable rule set size is being placed. The data set used is the *Cleveland heart disease database*, available from the UCI machine learning database repository (Murphy & Aha 1995).

10.2 Data Material

The data set used in the following experiment consists of information recorded from 303 consecutive patients referred to the Cleveland Clinic for coronary angiography between May 1981 and September 1984. For each patient, the results from several routine evaluations and tests in addition to three non-invasive tests as part of a research protocol, were recorded – in all 13 parameters. For details regarding the data collection procedure, see Detrano, Yiannikas, Salcedo, Rincon, Go, Williams & Leatherman (1984) and Detrano, Janosi, Steinbrunn et al. (1989). In addition to the clinical data and test results, the presence of coronary artery disease (the angiographic variable of the presence of a $> 50\%$ diameter narrowing) was recorded. The data set was used by Detrano et al. (1989) for logistic regression of a new discriminant function model predicting the disease probabilities of new patients. The model was tested on three different data sets from sites in Hungary, USA and Switzerland, and compared to the CADENZA system (Diamond, Staniloff, Forrester, Pollock & Swan 1983). For each data set, the accuracy of the two algorithms was plotted against the decision threshold. Using a range of thresholds to compare the accuracy makes the comparison more independent of prevalence, similarly to ROC analysis. The accuracy was generally slightly higher for the discriminant function, and it was concluded that discriminant functions are clinically useful for providing coronary disease probabilities. For the 0.5 threshold value, the accuracies for the discriminant function on the three data sets were 77%, 79% and 81%.

The Cleveland data has also been used as a combined training/testing set. Gennari, Langley & Fisher (1989) report an average accuracy of 78.9% with CLASSIT, a conceptual clustering system.

10.2.1 Attributes

The 13 attributes, in addition to the correct diagnosis, in the Cleveland data set are summarized in Tables 10.1 (non-numerical attributes) and 10.2 (numerical attributes).

Attribute	Description	Value	Distribution
sex	Sex	Female	32.0%
		Male	68.0%
cp	Chest pain type	Typical angina	7.6%
		Atypical angina	16.5%
		Non-anginal pain	28.4%
		Asymptomatic	47.5%
fbs	Fasting blood sugar > 120 mg/dl	False	85.1%
		True	14.9%
restecg	Resting electrocardiographic results	Normal	49.8%
		ST-T abnormality	1.3%
		LV hypertrophy	48.8%
exang	Exercise induced angina	No	67.3%
		Yes	32.7%
slope	The slope of the peak exercise ST segment	Upsloping	46.9%
		Flat	46.2%
		Downsloping	6.9%
thal	Exercise thallium scintigraphy	Normal	57.3%
		Fixed defect	6.5%
		Reversible defect	36.2%
disease	Diagnosis of heart disease	No	54.1%
		Yes	45.9%

Table 10.1: Non-numerical attributes from the Cleveland data set.

disease refers to the presence of heart disease in the patient, the angiographic disease status. “Yes” indicates > 50% narrowing¹. The value “ST-T abnormality” for the **restecg** attribute corresponds to having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV). The value “LV hypertrophy” corresponds to showing probable or definite left ventricular hypertrophy by Estes’ criteria.

¹The data base actually contains an attribute discriminating between four “presence” classes and one “absence” class. The previous experiments briefly discussed above have concentrated on distinguishing “presence” values from “absence” values; creating a binary decision problem by collapsing the “presence” classes into one class. This approach is also taken in the current experiment.

Attribute	Description	Unit	Mean	Std. dev.	Median
age	Age	years	54.4	9.0	56.0
trestbps	Resting blood pressure on admission to the hospital	mm Hg	131.7	17.6	130.0
chol	Serum cholestoral	mg/dl	246.7	51.8	241.0
thalach	Maximum heart rate achieved		149.6	22.9	153.0
oldpeak	ST depression induced by exercise relative to rest		1.0	1.2	0.8
ca	Number of major vessels colored by fluoroscopy		0.7	0.9	0.0

Table 10.2: Numerical attributes from the Cleveland data set.

10.3 Preprocessing

The Cleveland data set constitutes a decision system $\mathcal{A} = (U, \{C \cup \{disease\}\})$ where the universe U is the set consisting of the 303 patients, the set of condition attributes C consists of the 13 numerical and non-numerical attributes (excluding the `disease` attribute) and the decision attribute is `disease`. Preprocessing was needed to prepare the data for the inductive learning step.

10.3.1 Missing Values

Of the 303 patients, data were only missing for one attribute for each of 6 patients. For four of the patients data were missing for the `ca` attribute, for two of the patients data were missing for the `thal` attribute. Because of the small number of objects with missing values, the problem with missing values was solved by removing these patients from the initial data set, resulting in the decision system \mathcal{A}^R with an universe of 297 patients.

10.3.2 Splitting and Discretizing

As discussed in Chapter 4, the data set must be split into (at least) three subsets for learning and evaluation purposes. To avoid the properties of a particular split to make a substantial impact on the results, the whole experiment was repeated for three different splits. The data set contains continuous attributes which need to be discretized. Because discretization is a part of the training phase, the training set – instead of the complete data set – must be used by the discretization algorithm to find discretization intervals. Therefore, discretization must be done separately for each split.

The following procedure was repeated for i equal to 1, 2, and 3: The decision system \mathcal{A}^R was split into a testing set \mathcal{A}_{V_i} and a training set \mathcal{A}_{T_i} , containing $1/3$ and $2/3$ of the objects in \mathcal{A}^R respectively. Then, \mathcal{A}_{T_i} was used as input to a discretization algorithm based on boolean reasoning (Nguyen & Skowron 1995). The discretization algorithm split the value set of each numerical attribute into a finite set

of intervals, based on the information in the training set. These intervals were then used to create two new decision systems $\mathcal{A}_{T_i}^D$ and $\mathcal{A}_{V_i}^D$, by replacing the values for numerical attributes with symbols representing the intervals that the values lie in, for the training set and the testing set respectively. Then, $\mathcal{A}_{T_i}^D$ was again split into two equally sized sets, the learning set $\mathcal{A}_{L_i}^D$ and the hold-out set $\mathcal{A}_{H_i}^D$. A summary of the splits is shown in Table 10.3.

\mathcal{A}^R (297)					
		$\mathcal{A}_{T_i}^D$ (199)			
	Split i	$\mathcal{A}_{V_i}^D$ (98)	$\mathcal{A}_{L_i}^D$ (100)		$\mathcal{A}_{H_i}^D$ (99)
46.13% (137)	1	47.96% (47)	45.23% (90)	47.00% (47)	43.43% (43)
	2	36.75% (36)	50.75% (101)	45.00% (45)	56.57% (56)
	3	45.92% (45)	46.23% (92)	47.00% (47)	45.46% (45)

Table 10.3: Partitioning of the data. The table displays the prevalence – the percentage (and number) of patients with positive diagnosis `disease = "Yes"` – in each table (with table cardinality).

10.4 Initial Learning

For each of the three learning sets, object-related dynamic reducts modulo the decision attribute were computed by sampling 10 different subtables of each dimension from 50% to 90% in 10% steps – a total of 50 subtables. A summary of the reducts and the corresponding rule sets, called dyn_i for split i , is shown in Table 10.4.

Rule set	Reducts	Rules
dyn_1	1504	6309
dyn_2	1752	8248
dyn_3	1335	6292

Table 10.4: Initial rule sets and reducts

The goal of the experiment is pruning these rule sets down as much as possible, while retaining classifier performance.

10.5 Rule Quality

Similarly to the acute appendicitis experiment, the quality-based rule filtering algorithm (Figure 7.1) was run with *RUL* in turn equal to each of dyn_1 , dyn_2 and dyn_3 and *quality* equal to each of the quality functions listed on page 76. The *performance* parameter to the filtering algorithm was set to $AUC_{\mathcal{A}_{H_i}^D}$; the AUC evaluated on the hold-out set corresponding to the rule set. The classifier scheme outlined in Section 3.6 was used to classify the hold-out set. In the case of an empty set of firing rules for an object, $\phi(x)$ (the certainty of a positive diagnosis) was set

to the a priori probability of a positive decision; the positive decision rate from the learning set (see Table 10.3).

Below, some properties related to the computed quality measures are presented. The actual rule filtering is discussed in Section 10.7.

10.5.1 Quality Distributions

Distribution plots and summary similar to those for the acute appendicitis experiment in the previous chapter are presented in Appendix B.2 and Table 10.5, respectively.

The only quality formula that produced undefined quality values was the J-measure. Because of the large number of rules with undefined quality compared to the cardinalities of unfiltered rule sets, the rules with undefined quality were removed from the initial rule sets before filtering (the second approach in Section 7.3.2).

10.6 Genetic Computations

The genetic rule filtering algorithm in Figure 8.3 was run with RUL in turn equal to dyn_1 , dyn_2 and dyn_3 . In this section, parameter settings and results related to the operation of the genetic algorithm are presented. The rule filtering results are presented in Section 10.7.

10.6.1 Parameter Settings

Fitness Function and Initialization

performance and *complexity* for the fitness function were computed in the same way as for the acute appendicitis experiment (see Section 9.5), relative to the rule sets dyn_i .

The criteria used for setting the performance bias for the acute appendicitis experiment in Section 9.5.1 was considered to be reasonable for the current experiment also. The biases were thus calculated from Equation 9.1, and are shown in Table 10.6.

The cutoff value was set to 50; higher than the one for the acute appendicitis experiment because of larger rule sets.

The initialization rate p for each split are calculated from Equation 8.10, and shown in Table 10.6.

General Parameters

The general parameters to the genetic algorithm are shown in Table 10.7. See Sections 8.5 and 9.5.1 for description of, and comments on, the parameters.

Initial experiments showed that the genetic algorithm performs best for large rule sets when it is run with a small generation gap. Therefore, the gap was set to 50 generations in this experiment.

Quality measure	Rule set	Undefined	Best	Count	
				Avg (all)	Avg (100)
Michalski ($\mu = 0$)	<i>dyn</i> ₁	0	1	87.63	63.09
	<i>dyn</i> ₂	0	1	117.83	82.48
	<i>dyn</i> ₃	0	1	86.19	62.92
Michalski ($\mu = 0.5$)	<i>dyn</i> ₁	0	1	17.92	1.85
	<i>dyn</i> ₂	0	1	26.44	1.93
	<i>dyn</i> ₃	0	1	18.78	1.72
Michalski ($\mu = 0.8$)	<i>dyn</i> ₁	0	1	17.77	22.33
	<i>dyn</i> ₂	0	1	25.30	30.47
	<i>dyn</i> ₃	0	1	18.78	25.73
Torgo	<i>dyn</i> ₁	0	1	17.77	22.08
	<i>dyn</i> ₂	0	1	23.91	29.9
	<i>dyn</i> ₃	0	1	18.78	25.54
Brazdil	<i>dyn</i> ₁	0	1	17.77	1.87
	<i>dyn</i> ₂	0	1	23.91	1.96
	<i>dyn</i> ₃	0	1	18.78	1.82
Pearson	<i>dyn</i> ₁	0	1	24.98	1.97
	<i>dyn</i> ₂	0	1	24.99	1.96
	<i>dyn</i> ₃	0	1	18.95	1.81
J	<i>dyn</i> ₁	2069	1	13.42	2.36
	<i>dyn</i> ₂	2850	1	17.30	2.01
	<i>dyn</i> ₃	2434	1	12.90	1.97
Cohen	<i>dyn</i> ₁	0	1	21.61	3.57
	<i>dyn</i> ₂	0	1	27.40	7.73
	<i>dyn</i> ₃	0	1	22.88	3.36
Coleman	<i>dyn</i> ₁	0	2069	29.21	29.23
	<i>dyn</i> ₂	0	2850	41.87	40.81
	<i>dyn</i> ₃	0	2434	31.15	32.38
C1	<i>dyn</i> ₁	0	1	17.77	22.98
	<i>dyn</i> ₂	0	3	23.77	32.14
	<i>dyn</i> ₃	0	1	18.78	26.67
C2	<i>dyn</i> ₁	0	1	17.77	22.33
	<i>dyn</i> ₂	0	1	24.26	30.31
	<i>dyn</i> ₃	0	1	18.84	13.91
Kononenko	<i>dyn</i> ₁	0	882	29.94	29.66
	<i>dyn</i> ₂	0	1295	41.24	45.27
	<i>dyn</i> ₃	0	1260	30.84	33.57

Table 10.5: Breakdown of the quality value distribution. The table displays the number of rules with undefined quality, the number of rules with the (same) best quality (leaving out the rules with undefined quality), the average number of rules over all the quality values and the average number of rules over the 100 best quality values (again leaving out the rules with undefined quality).

Rule set	Performance bias	Initialization Rate
dyn_1	0.0277	0.0040
dyn_2	0.0214	0.0030
dyn_3	0.0278	0.0040

Table 10.6: Performance biases and Initialization Rates

Parameter	Value
Use scaling?	Yes
ΔT	0.02
Minimum T	1.45
Maximum T	6.45
n -point crossover	1
Crossover probability p_c	0.3
n -point mutation	1
Mutation probability p_c	0.05
Initial population size	400
Use elitism?	Yes
Sample with replacement?	Yes
Keep list generation gap g_k	50
Average fitness generation gap g_f	50

Table 10.7: Parameter settings

10.6.2 GA Performance

Here, results illustrating the genetic algorithm's ability to solve an optimization problem, similar to those in Section 9.5.2, are presented. See that section for general comments. Key statistics are shown in Table 10.8, and the curves showing the fitness functions and the change rates are shown in Figure 10.1 and 10.2, respectively.

Rule set	AUC	Generations	Best fitness		
			Value	Cardinality	AUC
dyn_1	0.9383	1817	0.9970	13	0.9649
dyn_2	0.9336	479	0.9969	16	0.9448
dyn_3	0.8926	1719	0.9961	14	0.9370

Table 10.8: Genetic algorithm statistics, showing the AUC on the hold-out sets for the unfiltered rule set and for the rule set with best fitness after termination.

The GA runs in this experiment used few generations compared to the acute appendicitis experiment, because of the comparatively low generation gap.

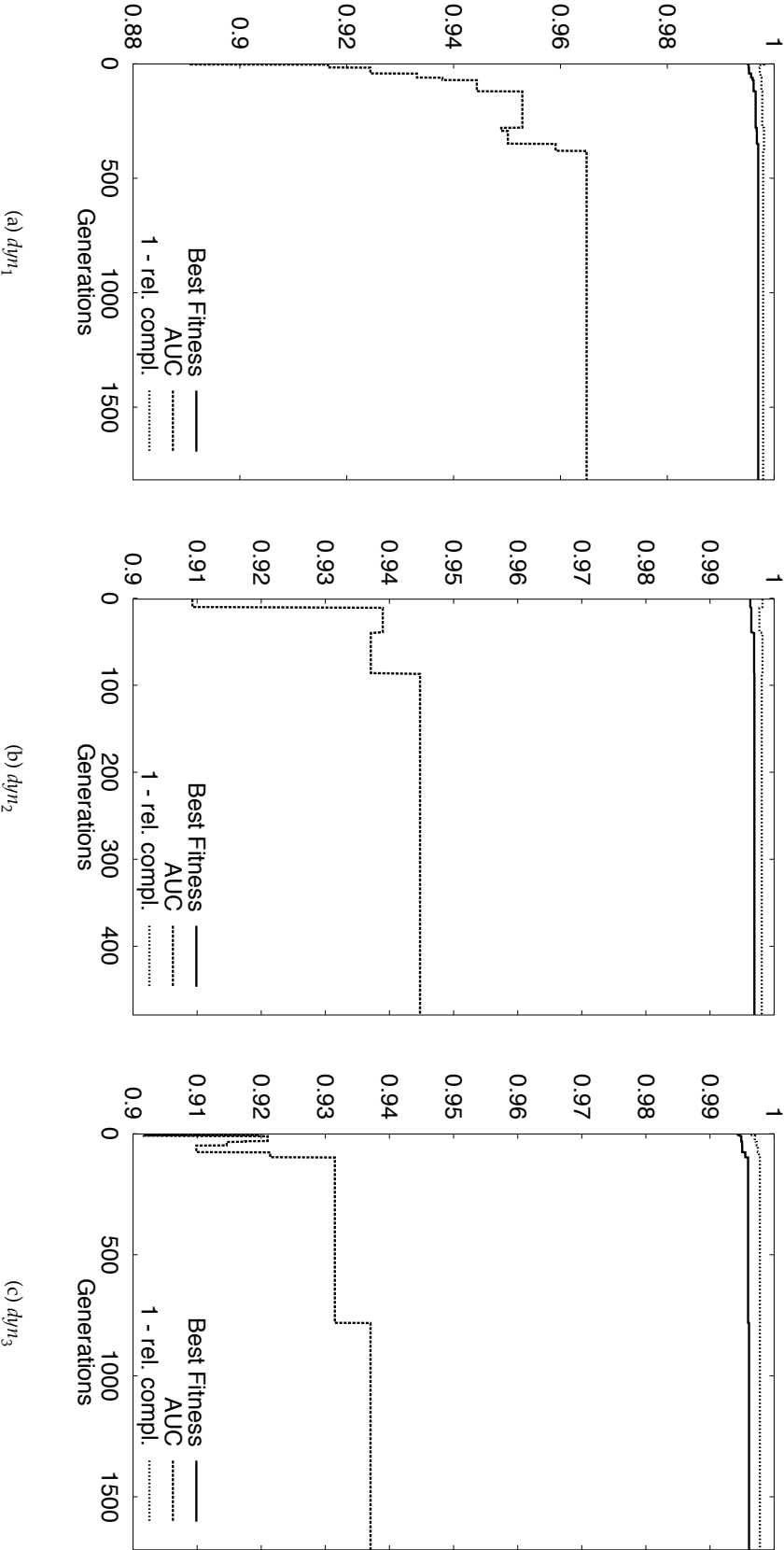


Figure 10.1: Performance, complexity and fitness

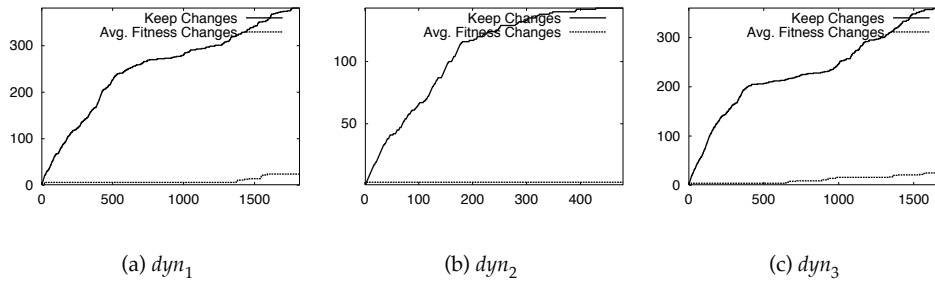


Figure 10.2: The number of generations that the keep list or the average fitness changed.

10.7 Rule Filtering

In this section, the rule filtering results are presented and discussed.

10.7.1 Model Selection

As discussed in Section 9.6.1, which model to select from a range of filtered models is a practical problem. In the current section, two different *ad-hoc* model selection methods are used with the quality filtering algorithm, and one with the genetic filtering algorithm. The following questions are considered:

- Which one of the two selection methods for the quality filtering algorithm gives models with best performance?
- Do the small filtered models perform as well as the unfiltered models?
- Which (if any) of the quality formulas performs best?
- Can we find better small models with the genetic algorithm than with the best quality formula?

Quality Filtering

The graphs computed by the quality filtering algorithm are displayed in Figure 10.3, showing the relationship between rule set size and AUC computed on the hold-out sets for the various quality formulae. In the graphs for the J-measure, the rules with undefined quality were removed before filtering.

The main emphasis of the current experiment is on filtering down the rule sets while retaining classifier performance, rather than on finding very small rule sets (the approach taken in the acute appendicitis experiment in the previous chapter). All the graphs in Figure 10.3 exhibit nice overall monotony, although with varying behavior in the parts corresponding to very small rule sets. The second consideration when selecting models for quality filtering in Section 9.6.1, that the graph should not fall too early, is not equally important in the current experiment, because no upper limit on the size of the usable models is defined. For each of the

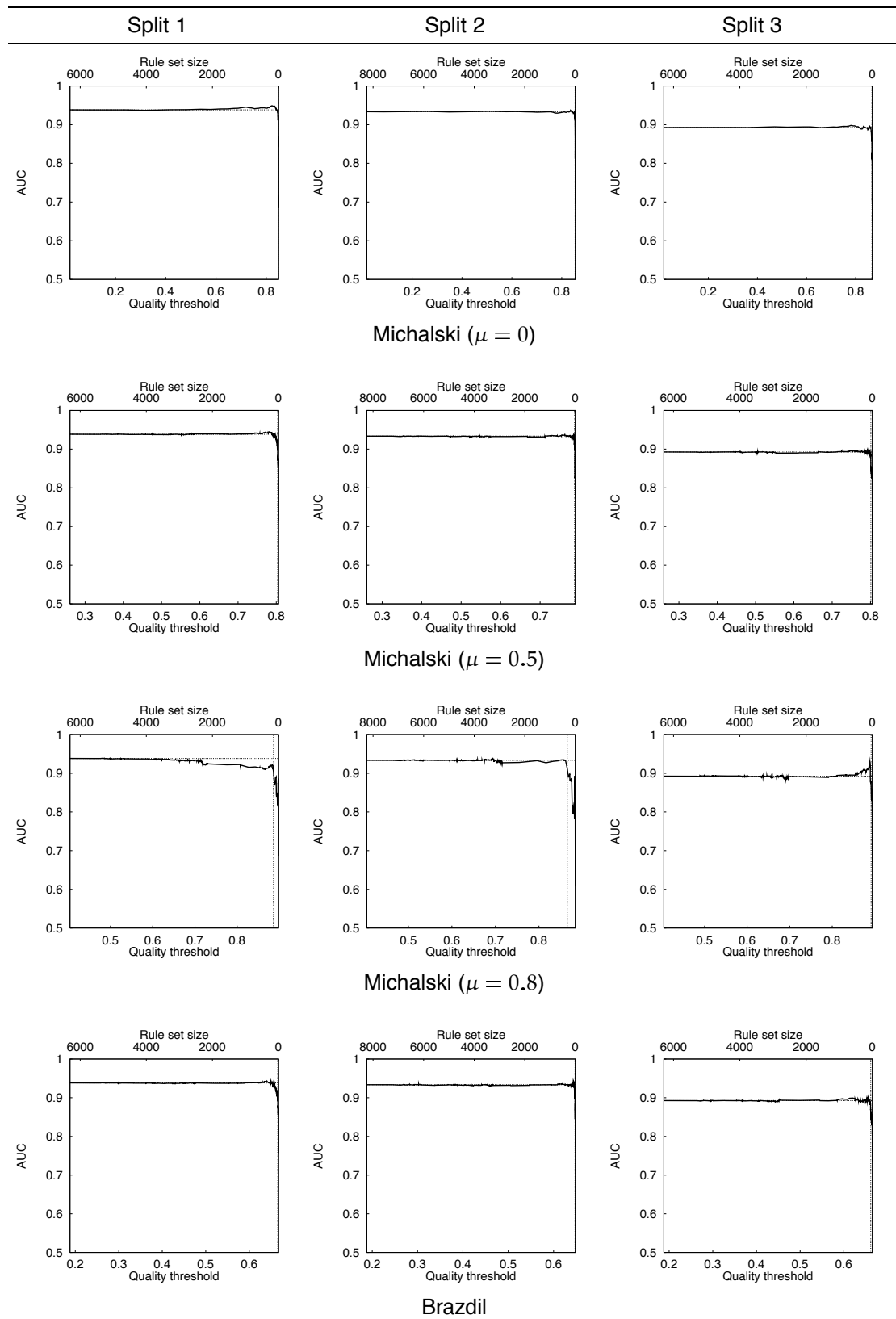


Figure 10.3: Quality Filtering. The horizontal dotted line represents the AUC for the unfiltered rule set. (Continues on page 107)

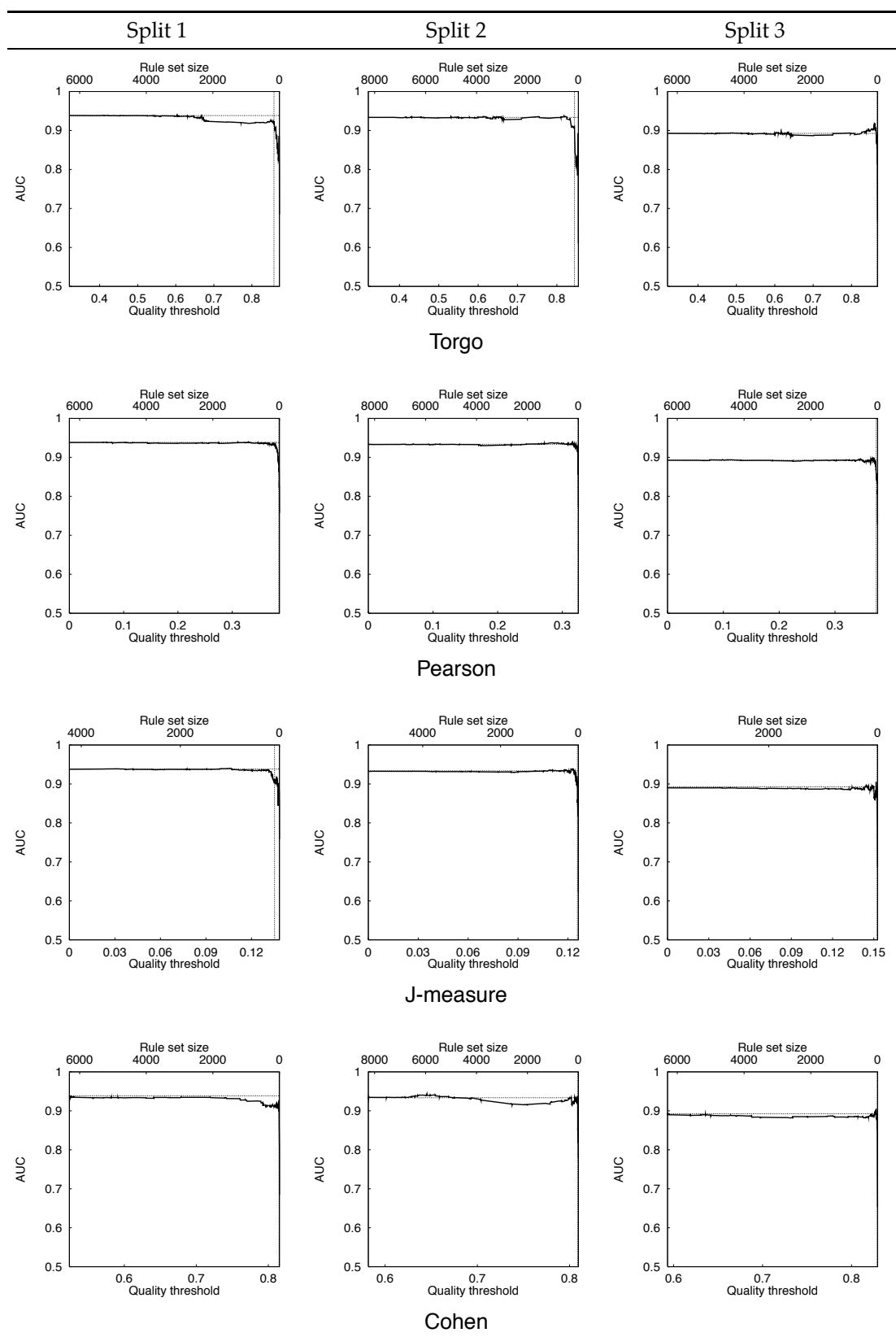


Figure 10.3: (Continued, continues on page 108)

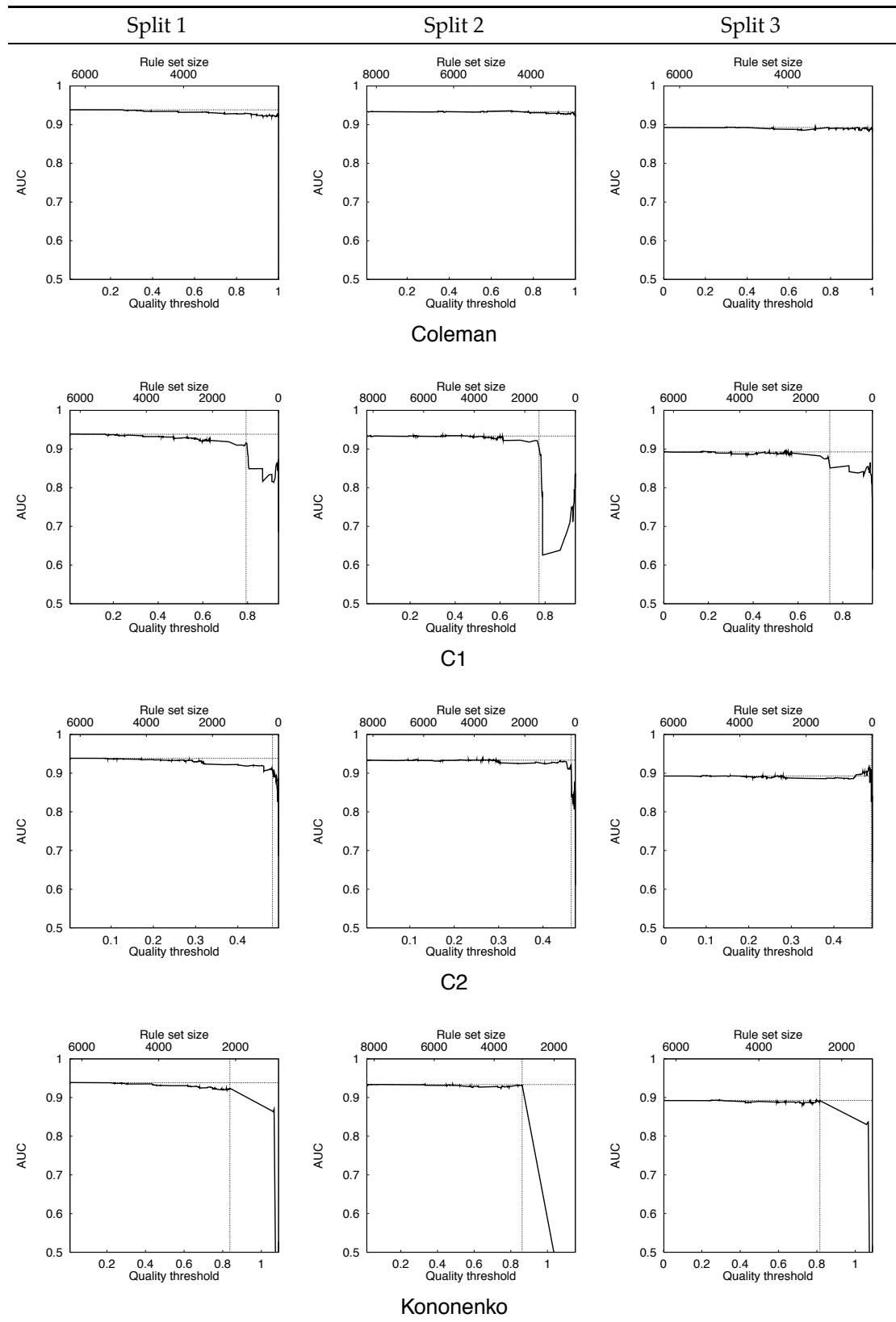


Figure 10.3: (Continued)

graphs in the figure, two different methods were used to select a model. The selected models are summarized in Table 10.10. The performance of the unfiltered rule sets on the hold-out sets is included for comparison in Table 10.9.

Visual Inspection The model at the knee-point or, generally, the smallest model in the smooth part of the graph was subjectively selected. The models selected by visual inspection are indicated by vertical dotted lines in Figure 10.3 (not distinguishable in all the graphs).

Significance Testing The selection method used for the acute appendicitis experiment. The model with the lowest cardinality without a significant performance drop compared to the unfiltered model was generally selected. Significant differences in performance were tested using Hanley / McNeil's hypothesis test (Section 4.5.2), generally with a significance level greater than 5%. The p-values are one-tailed and were computed using the Pearson correlation measure.

Rule set	Size	AUC (SE)
dyn_1	6309	0.9383 (0.0269)
dyn_2	8248	0.9336 (0.0252)
dyn_3	6292	0.8926 (0.0346)

Table 10.9: Unfiltered rule sets evaluated on the hold-out sets

Genetic Filtering

Graphs similar to those computed by the quality filtering algorithm were plotted from the keep-lists at the termination of the genetic algorithm, and are shown in Figure 10.4.

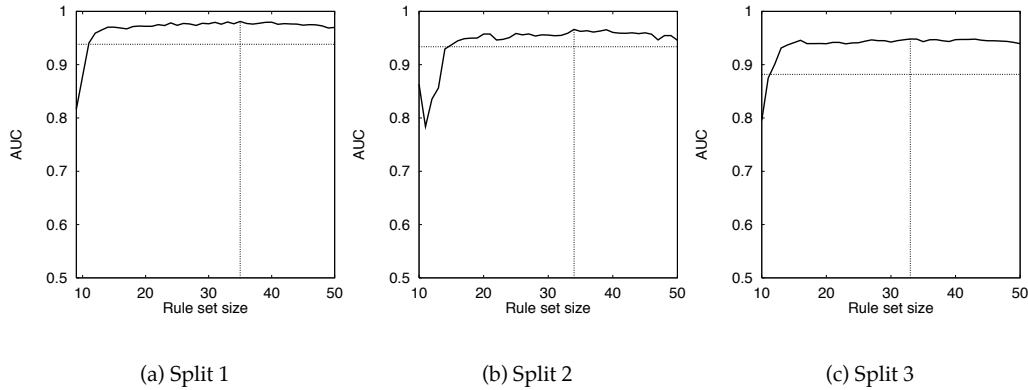


Figure 10.4: Genetic filtering. The horizontal dotted lines represent the AUCs computed on the holdout sets for the unfiltered rule sets.

Since the goal in the current experiment is not to find very small models, as were the case in the acute appendicitis experiment, the models with *highest performance*

Formula	Split	Significance Level			Visual Inspection				
		Threshold	Size	AUC (SE)	p-value	Threshold	Size	AUC (SE)	p-value
Michalski ($\mu = 0$)	1	0.6596	9	0.9140 (0.0316)	0.1673	0.3830	128	0.9483 (0.0247)	n/a*
	2	0.6667	12	0.9192 (0.0279)	0.1827	0.6222	15	0.9367 (0.0246)	n/a*
	3	0.6170	15	0.8603 (0.0391)	0.1188	0.5094	52	0.8934 (0.0344)	n/a*
Michalski ($\mu = 0.5$)	1	0.7021	27	0.8981 (0.0343)	0.0694	0.6796	44	0.9190 (0.0307)	0.1631
	2	0.6899	34	0.9238 (0.0271)	0.2641	0.6626	60	0.9358 (0.0248)	n/a*
	3	0.6868	47	0.8609 (0.0390)	0.0781	0.6829	50	0.8844 (0.0358)	0.3332
Michalski ($\mu = 0.8$)	1	0.8302	155	0.9059 (0.0330)	0.0576	0.8270	172	0.9207 (0.0304)	0.1710
	2	0.8218	322	0.9124 (0.0291)	0.1531	0.8182	389	0.9331 (0.0253)	0.4906
	3	0.8511	39	0.8671 (0.0382)	0.1694	0.8410	77	0.9298 (0.0281)	0.0485*
Brazdil	1	0.5512	24	0.8976 (0.0344)	0.0771	0.4548	178	0.9361 (0.0274)	n/a
	2	0.5527	13	0.9105 (0.0294)	0.1238	0.5011	65	0.9406 (0.0238)	n/a*
	3	0.5254	50	0.8623 (0.0388)	0.1024	0.4746	126	0.8965 (0.0340)	n/a*
Torgo	1	0.7877	165	0.9238 (0.0298)	0.2251	0.7877	165	0.9238 (0.0298)	0.2251
	2	0.7864	154	0.9097 (0.0295)	0.1363	0.7713	458	0.9352 (0.0249)	0.4584
	3	0.8349	15	0.8605 (0.0391)	0.1490	0.8018	85	0.9187 (0.0302)	0.1063
Pearson	1	0.2627	19	0.8860 (0.0361)	0.0563	0.1443	135	0.9363 (0.0273)	n/a
	2	0.2424	21	0.9138 (0.0288)	0.1870	0.2049	43	0.9263 (0.0266)	0.3366
	3	0.2340	41	0.8541 (0.0399)	0.0730	0.1671	99	0.8930 (0.0345)	n/a*
J	1	0.0629	100	0.9086 (0.0325)	0.0550	0.0460	217	0.9363 (0.0273)	n/a
	2	0.0880	25	0.9074 (0.0299)	0.1328	0.0692	72	0.9309 (0.0258)	0.4372
	3	0.1193	10	0.8613 (0.0390)	0.1689	0.0822	69	0.8914 (0.0347)	0.4735
Cohen	1	0.7911	12	0.9159 (0.0312)	0.2171	0.7903	14	0.9257 (0.0295)	0.3213
	2	0.7840	28	0.9134 (0.0289)	0.2127	0.7347	280	0.9340 (0.0252)	0.4909*
	3	0.8000	10	0.8560 (0.0397)	0.1087	0.7591	59	0.9010 (0.0332)	n/a*
Coleman	1	1.0000	2069	0.9223 (0.0301)	n/a	1.0000	2069	0.9223 (0.0301)	n/a
	2	1.0000	2850	0.9221 (0.0274)	n/a	1.0000	2850	0.9221 (0.0274)	n/a
	3	1.0000	2434	0.8897 (0.0350)	n/a	1.0000	2434	0.8897 (0.0350)	n/a
C1	1	0.8905	983	0.9159 (0.0313)	0.1161	0.8905	983	0.9159 (0.0313)	0.1161
	2	0.8825	1440	0.9047 (0.0304)	0.0792	0.8810	1507	0.9215 (0.0275)	0.2396
	3	0.8988	1287	0.8564 (0.0396)	0.0570	0.8921	1339	0.8803 (0.0364)	n/a
C2	1	0.3837	185	0.9076 0.0327	0.0516	0.3606	447	0.9153 (0.0314)	n/a
	2	0.3818	165	0.9174 (0.0282)	0.2087	0.3818	165	0.9174 (0.0282)	0.2087
	3	0.4239	35	0.8630 (0.0388)	0.1456	0.3966	107	0.9150 (0.0309)	0.1287*
Kononenko	1	0.9159	2149	0.9240 (0.0298)	n/a	0.9159	2149	0.9240 (0.0298)	n/a
	2	0.8625	3068	0.9323 (0.0255)	n/a	0.8625	3068	0.9323 (0.0255)	n/a
	3	0.9159	2527	0.8920 (0.0347)	n/a	0.9159	2527	0.8920 (0.0347)	n/a

Table 10.10: Selected models for quality based rule filtering. The AUC values are computed relative to the hold-out sets. p-values are given relative to the unfiltered rule sets, and values marked with a “*” indicates that the AUC for the filtered set is higher than the AUC for the unfiltered set. The p-values are one-tailed, computed using Pearson correlation. For fields marked “n/a”, the p-value could not be computed because the corresponding look-up values for the Hanley/McNeil correlation table were outside the scope of the table.

in the three keep-lists were selected instead of those at the knee-points. The selected models are indicated by vertical dotted lines in Figure 10.4, and summarized in Table 10.11.

Split	Size	AUC (SE)
1	35	0.9811 (0.0149)
2	34	0.9662 (0.0178)
3	33	0.9482 (0.0242)

Table 10.11: Selected models for genetic rule filtering. AUC is computed on the holdout sets. The rule sets are the ones with highest AUCs in the keep-lists.

Evaluation

All selected models were evaluated on the testing sets, and the performances statistically compared to those of the unfiltered sets. Statistical significance was defined at the 5% significance level for the one-tailed test. The results are shown in Tables 10.12 and 10.13.

Method	Split	Size	AUC (SE)	p-value
Unfiltered	1	6309	0.8721 (0.0371)	
Unfiltered	2	8248	0.9048 (0.0357)	
Unfiltered	3	6292	0.9335 (0.0395)	
Genetic	1	35	0.8669 (0.0378)	0.4291
Genetic	2	34	0.8842 (0.0391)	0.2414
Genetic	3	33	0.8818 (0.0362)	0.0008

Table 10.12: Performance evaluation on the testing sets: genetic and unfiltered. AUC values are computed on the testing sets. p-values are relative to the unfiltered sets.

General observations include that the models selected by visual inspection generally are slightly (insignificantly) better than the models selected by significance level. The former models are more often slightly better than the unfiltered models, compared to only once for the latter models. Of course, the models selected by visual inspection are generally larger. None of the models were *significantly* better than the unfiltered models. Note that the models produced by the Coleman and Kononenko formulas are large because of high best counts (see Table 10.5).

The statistical analysis can be used to partly answer the questions stated at the start of this section.

- Generally, the models selected by *visual inspection* are slightly better than the models selected by significance level, although the difference between corresponding models for the two selection methods never is significant. The models selected by visual inspection had less often (insignificantly) poorer performance compared to the unfiltered sets, and more often significantly better performance. However, the models selected by visual inspection were generally much larger than those selected by significance level

Method	Split	Significance Level		Visual Inspection	
		AUC (SE)	p-value	AUC (SE)	p-value
Michalski ($\mu = 0$)	1	0.8690 (0.0375)	0.4532	0.8600 (0.0387)	n/a
	2	0.8887 (0.0384)	0.2308	0.9052 (0.0356)	n/a*
	3	0.8574 (0.0395)	0.0006	0.9109 (0.0316)	n/a
Michalski ($\mu = 0.5$)	1	0.8210 (0.0433)	0.0431	0.8490 (0.0401)	0.1468
	2	0.9037 (0.0359)	0.4795	0.9100 (0.0349)	n/a*
	3	0.8912 (0.0348)	0.0206	0.9055 (0.0326)	n/a
Michalski ($\mu = 0.8$)	1	0.8419 (0.0409)	0.0748	0.8567 (0.0391)	0.2249
	2	0.8772 (0.0401)	0.1266	0.8880 (0.0385)	0.2298
	3	0.9004 (0.0334)	0.0597	0.8824 (0.0361)	0.0110
Brazdil	1	0.8350 (0.0417)	0.1233	0.8628 (0.0383)	n/a
	2	0.8799 (0.0397)	0.1938	0.9057 (0.0355)	n/a*
	3	0.8994 (0.0335)	0.0422	0.9025 (0.0330)	n/a
Torgo	1	0.8521 (0.0397)	0.1748	0.8521 (0.0398)	0.1748
	2	0.8665 (0.0416)	0.0843	0.8962 (0.0371)	0.3467
	3	0.8883 (0.0353)	0.0364	0.8837 (0.0360)	0.0102
Pearson	1	0.8333 (0.0419)	0.1595	0.8667 (0.0378)	n/a
	2	0.8696 (0.0412)	0.1424	0.8876 (0.0385)	0.2408
	3	0.8899 (0.0350)	0.0246	0.8994 (0.0335)	n/a
J	1	0.8573 (0.0390)	n/a	0.8653 (0.0380)	n/a
	2	0.8506 (0.0437)	0.0334	0.8867 (0.0387)	0.1996
	3	0.9086 (0.0320)	0.1405	0.8738 (0.0374)	0.0015
Cohen	1	0.8185 (0.0435)	0.0623	0.8212 (0.0432)	0.0685
	2	0.8707 (0.0410)	0.1679	0.9019 (0.0362)	0.4545
	3	0.8709 (0.0378)	0.0142	0.8954 (0.0342)	0.0198
Coleman	1	0.8717 (0.0371)	n/a	0.8717 (0.0371)	n/a
	2	0.9014 (0.0363)	n/a	0.9014 (0.0363)	n/a
	3	0.9229 (0.0295)	n/a	0.9229 (0.0295)	n/a
C1	1	0.8477 (0.0402)	0.1160	0.8477 (0.0402)	0.1160
	2	0.8837 (0.0391)	0.1815	0.8911 (0.0380)	0.2642
	3	0.9149 (0.0309)	0.1807	0.9168 (0.0306)	n/a
C2	1	0.8640 (0.0382)	n/a	0.8655 (0.0380)	n/a
	2	0.8403 (0.0450)	0.0117	0.8403 (0.0450)	0.0117
	3	0.9105 (0.0317)	0.1485	0.8927 (0.0346)	0.0179
Kononenko	1	0.8792 (0.0361)	n/a*	0.8792 (0.0361)	n/a*
	2	0.9010 (0.0364)	0.4231	0.9010 (0.0364)	0.4231
	3	0.9220 (0.0296)	n/a	0.9220 (0.0296)	n/a

Table 10.13: Performance evaluated on the testing sets: quality filtered rule sets, selected by significance level or visual inspection. p-values are given relative to the unfiltered rule sets, and values marked with a “*” indicate that the AUC for the filtered set is higher than the AUC for the unfiltered set. The p-values are one-tailed, computed using Pearson correlation. For fields marked “n/a”, the p-value could not be computed because the corresponding look-up values for the Hanley/McNeil correlation table were outside the scope of the table.

- Yes, it is possible to find very small models without significantly poorer performance than the unfiltered rule sets. The models selected by visual inspection for the formulas Michalski with $\mu = 0$ and $\mu = 0.5$, Brazdil and Pearson are all small and do not have significance performance drops for none of the splits. The three former formulas even give (insignificantly) *better* models for one split each. When selecting by significance level, only the Michalski formula with $\mu = 0.8$ gave models without significantly poorer performance for one or more of the splits.
- It is difficult to draw any valid conclusion regarding the relative performance of the quality formulas. In general, Michalski with $\mu = 0$ seems to perform slightly (insignificantly) better than the others, and also produces comparatively small models. Both the Brazdil and Pearson formulas also seem to perform well and give small models. The Coleman and Kononenko formulas produces models with high performance, but with a large number of rules. The Cohen formula produces models with few rules but relatively poor performance.
- No, there is not enough evidence to say that models found by the genetic algorithm perform better than those found by the quality based algorithm. Although there is no significant difference between the genetic models and e.g. the Michalski ($\mu = 0$) models, the latter perform slightly better.

ROC curves for the models selected for the genetic algorithm and the models selected by visual inspection for the quality formulas Michalski with $\mu = 0$, Brazdil and Pearson, measured on the testing sets, are shown in Figure 10.5. Properties of the classifiers corresponding to the points closest to (0,1) on the ROC curves are shown in Table 10.14.

Method	Split	τ	Sensitivity	Specificity	Accuracy
Genetic	1	0.520	0.8298	0.8039	0.8163
	2	0.344	0.8056	0.8871	0.8571
	3	0.420	0.80000	0.8679	0.8367
Michalski ($\mu = 0$)	1	0.472	0.7872	0.7647	0.7755
	2	0.432	0.7778	0.9032	0.8571
	3	0.396	0.8000	0.8491	0.8265
Brazdil	1	0.456	0.8298	0.7255	0.7755
	2	0.416	0.8056	0.8871	0.8571
	3	0.344	0.8000	0.8679	0.8367
Pearson	1	0.536	0.8085	0.7647	0.7857
	2	0.552	0.7778	0.8871	0.8469
	3	0.376	0.7556	0.8679	0.8163

Table 10.14: Classifier properties for τ corresponding to the points closest to (0, 1) on the ROC curves. Models for the quality formulas were selected by visual inspection.

10.8 Discussion

The same comments regarding experiment design as in the acute appendicitis experiment (Section 9.7 on page 9.7) apply to the experiment in this chapter. In ad-

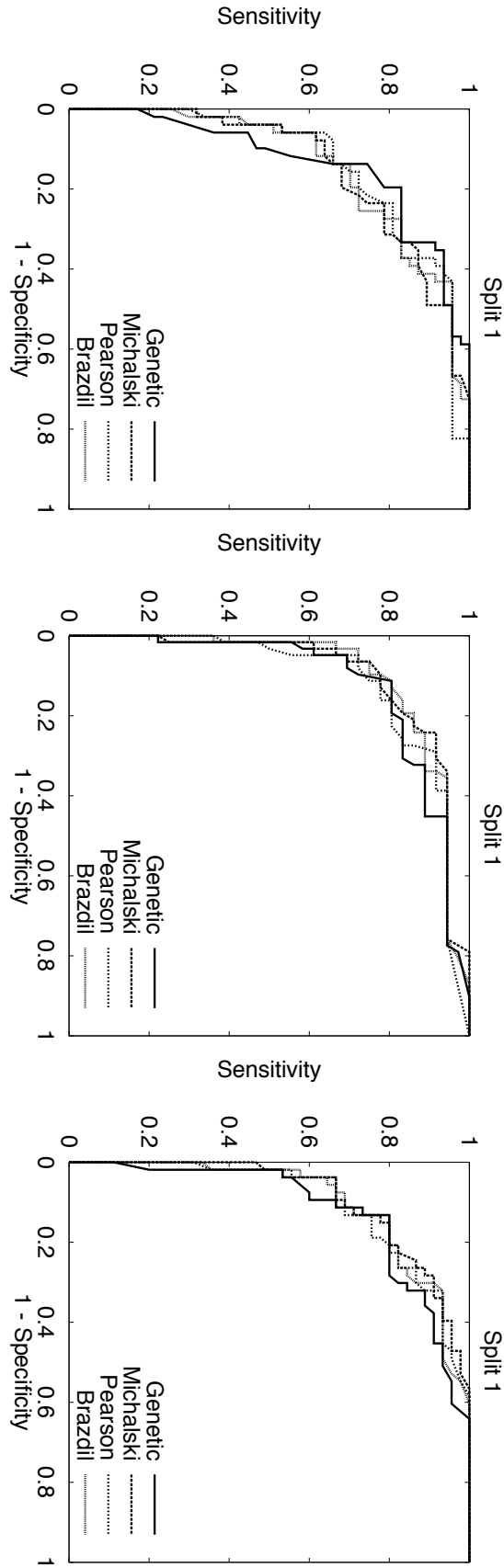


Figure 10.5: ROC curves for the selected classifiers on the testing sets. The models from the quality formulas were selected by visual inspection.

dition, the introduction of automatic discretization in the current experiment is an important factor when evaluating and comparing the experiments. The discretization step determines important aspects of the models that can be created by rough set methods, and may have strong influence on the results of experiments on real-world data. Alas, comparing alternative automatic or manual discretization methods were outside the time limits of the current work.

Discussion and Conclusion

11.1 Summary

A variety of machine learning algorithms implement a wide range of different model representations. When comparable predictive performance can be achieved, rule based models – such as those produced by methods from rough set theory – are usually preferred over e.g. artificial neural network or logistic regression models, because they are generally easily interpretable by humans. This facilitates the knowledge discovery aspect of machine learning. However in most cases, rule based models are too complex to be of any descriptive value. The focus of this thesis has been methods for finding the *best* of the *smallest* subsets of an induced rule set. Straightforward searching cannot be employed due to the large search space. Search with a genetic algorithm, generally a time consuming process, and search by finding a proper ordering of the rules, dramatically reducing the search space, have been implemented as general algorithms in ROSETTA accompanied by user interfaces (see Appendix A) for adapting the algorithms to specific problems.

11.2 Discussion

11.2.1 Quality Based Rule Filtering

Several of the twelve investigated quality formula variations exhibit similar properties. Particularly, some of the empirical ad-hoc formulas seem to imitate some of the statistically based formulas. The rule ordering by the Michalski formula with $\mu = 0.8$ is very similar to that by the C2 formula. Actually, for the three splits in the acute appendicitis experiment, the orderings only disagree at one point for the first ten rules, as seen in Figure 9.7; actually the ordering of the second and third rule in split 3 is switched for the two formulas. Also, the orderings of the ten first rules for the Michalski formula with $\mu = 0.5$ is very similar to that for the Brazdil formula (again see Figure 9.7).

Another observation is that in the experiments, the undefined counts and the best counts for the J and Coleman formulas, respectively, were equal. Actually, they will always be if the best quality value calculated by the Coleman formula is equal

to the maximum value 1. If $quality_{Coleman}(\alpha \rightarrow \beta) = 1$, then

$$\begin{aligned} \frac{|U| \cdot n_{\alpha,\beta} - n_{\alpha} \cdot n_{\beta}}{n_{\alpha} \cdot n_{\neg\beta}} &= 1 \\ |U| \cdot n_{\alpha,\beta} - n_{\alpha} \cdot n_{\beta} &= n_{\alpha} \cdot n_{\neg\beta} \\ &= n_{\alpha}(|U| - n_{\beta}) \end{aligned}$$

or

$$n_{\alpha} - n_{\alpha,\beta} = 0$$

giving

$$n_{\alpha,\neg\beta} = 0 \tag{11.1}$$

Equation 11.1 is exactly the condition for when the J-measure is undefined. The rules with undefined J-measure (and maximum Coleman quality) are intuitively *good* rules. Thus, rules with undefined quality should preferably *not* be removed when using the J-measure (alternative 1 in Section 7.3.2 should be used). In principle, this makes the Coleman formula and the J-measure of limited value when searching for the smallest models, because of the low resolution in the high end of the quality scale. It is surprising, however, that even though over fifty of the best rules are removed, the J-measure exhibits very good performance in the acute appendicitis experiment.

The experiments backs up one of the results from Bruha (1997), namely that there is no pronounced distinction between the performance in the class of empirical formulas and the class of statistically based formulas. One other result from Bruha (1997) is reversed; in the current study the Michalski formula seems to perform better with lower accuracy bias instead of the other way around as in Bruha's investigation. One reason for this is the different classification schemes employed. In one of the schemes used by Bruha, only the rule with best quality that fires for a given object is used to classify that object. Every object will thus be classified accurately. The voting scheme used in the current investigation, however, employs the same set of rules on every object. Rules with high coverage will potentially fire for a relative large proportion of the rules, and thus give good performance in the current classification scheme. Filtering with coverage only is the approach taken by Øhrn et al. (1998b).

11.2.2 Genetic Rule Filtering

In the acute appendicitis experiment with relatively small unfiltered models, the genetic algorithm performed slightly better than quality filtering – even with smaller models than those selected for in the quality filtering. In the Cleveland experiment with comparatively large rule sets, however, the rule sets selected for the genetic algorithm were of comparable size but generally exhibited slightly poorer performance than the quality filtered models. This is a fact although the genetic models for the first experiment were selected at the knee-points and those for the second experiment were selected as the best performers – a supposedly performance advantage in favor of the second experiment. One possible explanation is the much larger search space for the genetic algorithm in the second case. The only preliminary conclusion that can be drawn is that it seems like the genetic algorithm does not perform well on very large rule sets, compared to the quality based algorithm.

The genetic algorithm uses significantly more computation time to find small rule sets with comparable performance, than the quality based algorithm. This makes it unsuitable in contexts where “on-line” rule filtering is needed. In other contexts, where the rule filtering step is done only once and the running time is not important, genetic filtering may be employed and can often find better small models than the quality filtering scheme. The running times of the genetic algorithm is directly dependent upon the CPU speed and also partly upon the available computer memory. The faster the CPU the faster the population will converge, and the more memory the larger population size can be used – decreasing the number of generations needed for convergence.

Using a fitness function directly dependent upon classifier performance (Equation 8.3 on page 60) corresponds to comparing the raw performance measures of classifiers. Small differences between raw performance may however be random, and some kind of statistical hypothesis test should ideally be employed to compare performance in the fitness function. This is difficult both for efficiency reasons, and because the fitness function should operate on a continuous scale.

11.3 Conclusions

The results from both experiments are in favor of Hypothesis 1 (on page 3). In the experiments, submodels with significantly lower complexity and without statistically significant performance loss were found.

Hypothesis 2 is very hard to satisfy if taken literally. Referring to Figures 9.4, 9.7 and 10.3, none of the graphs are non-decreasing as a function of the number of rules, but some of them – i.e. Michalski with $\mu = 0$ – exhibit a general monotonically non-decreasing trend. The conclusion is that there exist functions creating orderings with *approximately* non-decreasing performance. Note that Hypothesis 2 is a condition but not a sufficiency for a proper rule filtering quality function.

The functions investigated here do not perform well enough to validate Hypothesis 3. In the acute appendicitis experiment, better models were found with the genetic algorithm. But generally, the best quality functions found almost as good models as (or better than) the genetic algorithm, and the preliminary conclusion is that the best quality functions generally seem to find very good submodels for a given cardinality.

In addition to the hypotheses from the introductory chapter, the following conclusions are made:

- The Michalski formula with $\mu = 0$ and the Pearson χ^2 statistic are recommended used for quality based rule filtering. These functions have shown generally good performance and good stability over a range of cardinalities.
- The J-measure is also recommended. If possible, the rules with undefined J-measure should not be removed but should be considered having maximal quality, although the J-measure seems to perform well even if these rules are removed.
- The Cohen, Coleman, and Kononenko formulas can not be recommended for quality based rule filtering, based on the results from the experiments.
- Relatively surprisingly, filtering by coverage only is better than any of the tested empirically based formulas combining coverage and accuracy.

- If the goal is to prune a large fraction of the rules and it is important to retain performance, the models for the quality filtering should be selected by visual inspection. If it is important to find very small models, while allowing small differences in predictive performance, models should be selected by significance level.
- The quality based filtering algorithm should be used instead of the genetic algorithm on very large rule sets (refer the sizes of the rule sets in the previous experiments). For smaller rule sets, if the particular application allows several hours of computation time and the target is rule sets with more than five rules, then the genetic algorithm should be used. Otherwise, quality based rule filtering using one of the formulas recommended above should be used.
- Rule-based models with 6-8 rules predicting acute appendicitis slightly better than surgeons, can be constructed. The performance difference is not always significant.

11.4 A Note on the Validity of the Experimental Results

The acute appendicitis data set used in the experiment may not be an ideal data set to base conclusions on. A recent¹ study by Øhrn (1999a) shows that small sets of so-called 1R rules perform well on this particular data set. 1R rules (Holte 1993) are very simple univariate rules. There is a danger that the existence of small, good sets of 1R rules may reduce the filtering algorithms to search methods for finding these sets (which can be very easily extracted by other means). In addition, the existence of small sets of 1R rules rules out the potential conclusion that there does not exist small, good rule sets at all. On another data set this may be a possible conclusion. Inspection of the rule sets generated by the filtering algorithms shows, however, that they in fact do not consist entirely of 1R rules; showing that other – possibly interesting – small models also can be found. Experiments on two data sets only are also a relatively weak foundation for general conclusions. General conclusions should be drawn from experiments on a range of data sets from different domains and with different learning algorithms. Particularly, even larger rule sets than those considered in the Cleveland experiment should be tested. The conclusions given herein must therefore be seen as preliminary propositions based upon the two experiments in the previous chapters. On the other hand; there is no significant factors indicating that the results should not be valid in the general case.

Also note that the results are relative to the particular decision rule classifier scheme used (see Section 3.6).

11.5 Limitations

Rule filtering can be seen as the removal of rules originated from observations that do not agree with the general “trends” in the data. However, the simple rule filtering schemes considered in this work do not discern between random noise and rare but potentially important correct observations. A rule that only help explain small parts of the data is removed. One solution to this problem is to use a skew

¹Not available before after the experiment in the current work was finished.

distribution of observations in the learning set, i.e. by “multiplying up” the class of objects of interest.

11.6 Further Work

The following are some of the steps that should be taken to try to confirm and extend the results presented in this thesis. Experimental work that should be done include:

- The quality based algorithm and the genetic rule filtering algorithm should be compared for filtering rule sets significantly larger than those used in the two experiments, i.e. with more than 100,000 rules. An interesting hypothesis is that quality based filtering outperforms genetic filtering for very large rule sets. A thorough experiment including statistical validation should be carried out.
- An extensive experiment involving several different data sets should be designed and carried out. Statistical validation of the results over several data sets should be carried out.
- Results for the genetic algorithm should be compared for other parameter settings than the parameters used in the experiments. The impact of e.g. the mutation fraction on the convergence rate should be investigated.
- Several extensions of the genetic algorithm should be considered; for example the inclusion of a transposition operator for potentially increasing the variety of the population.
- Other sophisticated search techniques, like simulated annealing, should be tried out and compared to the other algorithms.
- Filtered rule sets should be statistically compared to small 1R rule sets generated from the same data.

General approaches to pruning rule based models that should be considered include:

- Algorithms for rule filtering using complexity measures with higher resolution, for example descriptor count, should be considered. Such measures would give a more real image of a model’s complexity.
- Methods for filtering of the data input to the learning algorithm should be investigated, in order to focus on specific features such as general trends in the data.
- Entropy-based methods for rule pruning, such as those used by Quinlan (1986), should be considered for filtering sets of propositional rules.
- More sophisticated methods for visualization of rule based models should be investigated, including the use of domain information to simplify rule sets.

Bibliography

- Bazan, J. G. (1998), A comparison of dynamic and non-dynamic rough set methods for extracting laws from decision tables, in L. Polkowski & A. Skowron, eds, 'Rough Sets in Knowledge Discovery 1: Methodology and Applications', number 18 in 'Studies in Fuzziness and Soft Computing', Physica-Verlag, Heidelberg, Germany, chapter 17, pp. 321–365.
- Bazan, J. G., Skowron, A. & Synak, P. (1994), Dynamic reducts as a tool for extracting laws from decision tables, in 'Proc. International Symposium on Methodologies for Intelligent Systems', number 869 in 'Lecture Notes in Artificial Intelligence', Springer-Verlag, pp. 346–355.
- Bergadeno, F. et al. (1988), Measuring quality of concept descriptions, in 'EWSL-88', Glasgow.
- Bishop, Y. M. M., Fienberg, S. E. & Holland, P. W. (1991), *Discrete Multivariate Analysis: Theory and Practice*, MIT Press, Cambridge, Massachusetts.
- Brachman, R. J. & Anand, T. (1996), The process of knowledge discovery in databases: A human-centered approach, in U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth & R. Uthurusamy, eds, 'Advances in Knowledge Discovery and Data Mining', AAAI Press / The MIT Press, pp. 37–57.
- Brazdil, P. & Torgo, L. (1990), Knowledge acquisition via knowledge integration, in 'Current Trends in Knowledge Acquisition', IOS Press.
- Breiman, L., Friedman, J. H., Olshen, R. A. & Stone, C. J. (1984), *Classification and Regression Trees*, Wadsworth International Group.
- Brown, F. M. (1990), *Boolean Reasoning: The Logic of Boolean Equations*, Kluwer Academic Publishers, Dordrecht, The Netherlands.
- Bruha, I. (1997), Quality of decision rules: Definitions and classification schemes for multiple rules, in G. Nakhaeizadeh & C. C. Taylor, eds, 'Machine Learning and Statistics, The Interface', John Wiley and Sons, Inc., chapter 5.
- Brunk, C. A. & Pazzani, M. J. (1991), An investigation of noise-tolerant relational concept learning algorithms, in L. Birnbaum & G. Collins, eds, 'Proceedings of the 8th International Workshop on Machine Learning', Morgan Kaufmann, pp. 389–393.
- Carlin, U. (1998), Mining medical data with rough sets, Master's thesis, Norwegian University of Science and Technology.
- Carlin, U., Komorowski, J. & Øhrn, A. (1998), Rough set analysis of medical datasets in a case of patients with suspected acute appendicitis, in 'Proc. ECAI'98 Workshop on Intelligent Data Analysis in Medicine and Pharmacology (IDAMAP'98)', pp. 18–28.

- Clark, P. & Niblett, T. (1989), 'The CN2 induction algorithm', *Machine Learning* **3**, 261–283.
- Cohen, J. (1960), 'A coefficient of agreement for nominal scales', *Educ. Psychol. Meas.* **20**, 37–46.
- Cohen, W. W. (1993), Efficient pruning methods for separate-and-conquer rule learning systems, in 'IJCAI-93'.
- Detrano, R., Janosi, A., Steinbrunn, W. et al. (1989), 'International application of a new probability algorithm for the diagnosis of coronary artery disease', *The American Journal of Cardiology* **64**, 304–310.
- Detrano, R., Yiannikas, J., Salcedo, E., Rincon, G., Go, R., Williams, G. & Leatherman, J. (1984), 'Bayesian probability analysis: a prospective demonstration of its clinical utility in diagnosing coronary disease', *Circulation* **69**, 541–547.
- Diamond, G., Staniloff, H., Forrester, J., Pollock, B. & Swan, H. (1983), 'Computer-assisted diagnosis in the noninvasive evaluation of patients with suspected coronary artery disease', *JACC* **1**, 444–455.
- Esposito, F., Malerba, D. & Semeraro, G. (1993), Decision tree pruning as a search in the state space, in P. B. Brazdil, ed., 'Proceedings of the European Conference on Machine Learning (ECML-93)', Vol. 667 of *LNAI*, Springer Verlag, Vienna, Austria, pp. 165–184.
- Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P. & Uthurusamy, R. (1996), *Advances in Knowledge Discovery and Data Mining*, AAAI Press / The MIT Press.
- Fürnkranz, J. (1994), FOSSIL: A robust relational learner, in F. Bergadano & L. de Raedt, eds, 'Proceedings of the European Conference on Machine Learning', Vol. 784 of *LNAI*, Springer, Berlin, pp. 122–137.
- Fürnkranz, J. (1997), 'Pruning algorithms for rule learning', *Machine Learning* **27**, 139.
- Gennari, J. H., Langley, P. & Fisher, D. (1989), 'Models of incremental concept formation', *Artificial Intelligence* **40**, 11–61.
- Goldberg, D. E. (1989), *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, Mass.
- Halland, S., Åsberg, A. & Edna, T.-H. (1997a), 'Additional value of biochemical tests in suspected acute appendicitis', *European Journal of Surgery* **163**(7), 533–538.
- Halland, S., Åsberg, A. & Edna, T.-H. (1997b), 'Estimating the probability of acute appendicitis using clinical criteria of a structured record sheet: The physician against the computer', *European Journal of Surgery* **163**(6), 427–432.
- Hanley, J. A. & McNeil, B. J. (1982), 'The meaning and use of the area under a receiver operation characteristic (ROC) curve', *Radiology* **143**, 29–36.
- Hanley, J. A. & McNeil, B. J. (1983), 'A method for comparing the areas under receiver operating characteristic curves derived from the same cases', *Radiology* **148**, 839–843.
- Holland, J. H. (1975), *Adaptation in natural artificial systems*, University of Michigan Press, Ann Arbor.

- Holte, R. C. (1993), 'Very simple classification rules perform well on most commonly used datasets', *Machine Learning* **11**, 63–91.
- Kononenko, I. & Bratko, I. (1991), 'Information-based evaluation criterion for classifier's performance', *Machine Learning* **6**, 67–80.
- Kowalczyk, W. (1998), *Rough Data Modelling : a new technique for analyzing data*, Vol. 1 of *Studies in Fuzziness and Soft Computing*, Physica-Verlag, chapter 20, pp. 400–421.
- Michalewicz, Z. (1996), *Genetic Algorithms + Data Structures = Evolution Programs*, 3 edn, Springer, Berlin.
- Michalski, R. S. (1983), A theory and methodology of inductive learning, in R. S. Michalski, J. G. Carbonell & T. M. Mitchell, eds, 'Machine Learning, An Artificial Approach', Tioga Publishing, Palo Alto, chapter 4.
- Mitchell, M. (1996), *An Introduction to Genetic Algorithms*, MIT Press.
- Mollestad, T. (1997), A Rough Set Approach to Data Mining: Extracting a Logic of Default Rules from Data, PhD thesis, The Norwegian University of Science and Technology, Trondheim, Norway.
- Murphy, P. M. & Aha, D. W. (1995), *UCI Repository of Machine Learning Databases*, Machine-readable collection, Dept of Information and Computer Science, University of California, Irvine. [Available by anonymous ftp from `ics.uci.edu` in directory `pub/machine-learning-databases`].
- Nguyen, H. S. & Skowron, A. (1995), Quantization of real-valued attributes, in 'Proc. Second International Joint Conference on Information Sciences', Wrightsville Beach, NC, pp. 34–37.
- Øhrn, A. (1998), *ROSETTA: Technical Reference Manual*, draft version November 6 edn, The Norwegian University of Science and Technology, Trondheim, Norway.
- Øhrn, A. (1999a), Diagnosing acute appendicitis with very simple classification rules, Technical report, Dept. of Computer and Information Science, Norwegian University of Science and Technology (NTNU), Trondheim, Norway.
- Øhrn, A. (1999b), 'The Rosetta Homepage [url: http://www.idi.ntnu.no/~aleks/rosetta/](http://www.idi.ntnu.no/~aleks/rosetta/)'.
- Øhrn, A., Komorowski, J., Skowron, A. & Synak, P. (1998a), The design and implementation of a knowledge discovery toolkit based on rough sets: The ROSETTA system, in L. Polkowski & A. Skowron, eds, 'Rough Sets in Knowledge Discovery 1: Methodology and Applications', number 18 in 'Studies in Fuzziness and Soft Computing', Physica-Verlag, Heidelberg, Germany, chapter 19, pp. 376–399.
- Øhrn, A., Ohno-Machado, L. & Rowland, T. (1998b), Building manageable rough set classifiers, in 'Proc. AMIA Annual Fall Symposium', Orlando, FL, USA, pp. 543–547.
- Pawlak, Z. (1982), 'Rough sets', *International Journal of Information and Computer Science* **11**(5), 341–356.
- Pawlak, Z. (1984), On learning - a rough set approach, in A. Skowron, ed., 'Proceedings of the 5th Symposium on Computation Theory', Vol. 208 of *LNCS*, Springer, Zaborów, Poland, pp. 197–227.

- Pawlak, Z. (1991), *Rough Sets – Theoretical Aspects of Reasoning about Data*, Kluwer Academic Publishers, Dordrecht.
- Pawlak, Z. & Skowron, A. (1993), A rough sets approach to decision rules generation, Technical report, University of Warsaw.
- Pearl, J. (1978), 'On the connection between the complexity and credibility of inferred models', *Journal of General Systems* 4, 255–264.
- Provost, F., Fawcett, T. & Kohavi, R. (1998), The case against accuracy estimation for comparing induction algorithms, in 'Proceedings of the Fifteenth International Conference on Machine Learning (IMLC-98)', Madison, WI.
- Quinlan, J. R. (1986), 'Induction of decision trees', *Machine Learning* 1(1), 81–106.
- Quinlan, J. R. (1987), 'Simplifying decision trees', *International Journal of Man-Machine Studies* 27(3), 221–234.
- Quinlan, J. R. (1990), 'Learning logical definitions from relations.', *Machine Learning* 5(3), 239–266.
- Ripley, B. D. (1996), *Pattern Recognition and Neural Networks*, Cambridge University Press, Cambridge.
- Russell, S. & Norvig, P. (1995), *Artificial Intelligence: A Modern Approach*, Prentice-Hall, Englewood Cliffs, NJ, ISBN 0-13-103805-2, 912 pp., 1995.
- Salzberg, S. L. (1997), 'On comparing classifiers: Pitfalls to avoid and a recommended approach', *Data Mining and Knowledge Discovery* 1(3), 317–328.
- Skowron, A. (1993), Boolean reasoning for decision rules generation, in J. Komorowski & Z. W. Raś, eds, 'Proceedings of the 7th International Symposium on Methodologies for Intelligent Systems (ISMIS'93)', Vol. 689 of *LNAI*, Springer Verlag, Trondheim, Norway, pp. 295–305.
- Skowron, A. & Rauszer, C. (1991), The discernibility matrices and functions in information systems, in R. Slowinski, ed., 'Intelligent Decision Support – Handbook of Applications and Advances in Rough Set Theory', Kluwer Academic Publishers, pp. 331–362.
- Smyth, P. & Goodman, R. M. (1990), Rule induction using information theory, in G. Piatetsky-Shapiro & W. Frawley, eds, 'Knowledge Discovery in Databases', MIT Press, Cambridge, MA.
- Swets, J. A. (1988), 'Measuring the accuracy of diagnostic systems', *Science* 240, 1285–1293.
- Torgo, L. (1993), Controlled redundancy in incremental rule learning, in P. B. Brazdil, ed., 'Proceedings of the European Conference on Machine Learning (ECML-93)', Vol. 667 of *LNAI*, Springer Verlag, Vienna, Austria, pp. 185–195.
- Vinterbo, S. (1999), Finding minimal cost hitting sets: A genetic approach, Not yet published.

APPENDIX A

Implementation

A.1 Introduction

This appendix describes the implementation of the genetic and the quality based rule filtering algorithms in the data analysis toolkit ROSETTA.

A.2 Rosetta

ROSETTA (Øhrn et al. 1998a) is a toolkit for knowledge discovery and data analysis within the framework of rough set theory. The system is a result of a joint effort between the Norwegian University of Science and Technology and the University of Warsaw, Poland, and is currently available (Øhrn 1999b) as an application program running under Windows NT. It was mentioned in the introductory chapter that data mining is only one of the steps in the knowledge discovery process. The complete process consists of a number of steps, such as data selection, preprocessing, discretization, interpretation of the results, etc., and is not a strict pipeline but includes feedbacks between the sub steps. In addition, the selection of a particular tool for a particular step is specific for a particular KDD task. ROSETTA offers an experimental, interactive environment for data analysis, comprising most of the steps in the KDD process and offering a range of algorithms for each of the steps. ROSETTA has been successfully employed for data analysis in a variety of fields.

ROSETTA is implemented in the C++ programming language, and the design of the application is divided into a front-end and a kernel. The front-end constitutes a graphical user interface reflecting the contents of the kernel. The kernel is independent of the front-end, and can be used with other interfaces in future applications.

A.2.1 Kernel Architecture

See Øhrn et al. (1998a) for a general presentation of the ROSETTA design. The kernel is a C++ class library, and the main object dichotomy is into `structure` and `algorithm` objects. Different structure classes represent different types of data. An algorithm object is applied to a structure object to either mutate the latter or to

create a new structure object. For example, applying a learning algorithm (an algorithm object) to a decision table object (a structure) produces a set of rules (another structure). New algorithms are embedded in the class library by specialization of an existing (possibly virtual) class. All algorithms installed in the kernel are automatically available in the front-end.

A.3 The Genetic Algorithm

The implementation of a genetic algorithm by Vinterbo (1999) was already available in the ROSETTA library. While the main architecture of this algorithm could be re-used, considerable parts had to be re-implemented due to performance issues.

A.3.1 Architecture

The main architecture of the general genetic algorithm consists of an `engine` object maintaining a `population` object consisting of individual objects. The `engine` object iteratively applies a predefined set of `abstract operation` objects, such as `selection` and `mutation`, to the population, evaluates the population via an `evaluator` object using a `fitness` object, and updates a `keeper` object.

The genetic rule filtering algorithm takes a `rules` object as input, and returns the same rule set without altering it. The main result is a log file, updated by the `engine` object, showing statistics for each generation and the contents of the `keeper` object at the termination of the algorithm.

A.3.2 Representation

In the original implementation (Vinterbo 1999) of the genetic algorithm, the implementation of the `individual` class uses a fixed-length representation. An individual defined over a set of genes G use $|G|$ memory locations, no matter whether a particular gene is present in the individual or not. In other words, the individual represented by the string 00000000 uses as much memory as the individual represented by the string 11111111. This represents no problem in codings with relatively few genes, but in the coding of the rule filtering problem:

- The set of genes can be very large, up to hundreds of thousands of genes
- The individuals will generally be very sparse

For large rule sets, a fixed-length `individual` representation would both demand a large amount of memory and would lead to an unacceptable computational overhead due to the handling of the large data structures, in a population consisting of several hundreds individuals. Memory management in a genetic algorithm is important because the more memory used by an individual, the fewer individuals can be kept in a population and the slower the algorithm will find an acceptable solution. However, the individuals can generally be assumed to be quite sparse. If a rule set with thousands of rules is to be filtered down to, say, ten rules, individuals representing large rule sets will be heavily punished by the fitness function, leading to a rapid decrease in the average number of rules represented by

the individuals in the population. Because of this, a new implementation of the individual class was created. In this new implementation, each individual is represented by a dynamic-sized vector consisting of the indices to the “1”s in the corresponding bitstring. This way, an individual representing 5 rules from a set of 50,000 rules is represented by a vector of length 5 instead of 50,000 as in the original implementation. However, introducing a new individual class represented a problem, as the original architecture was not design with this type of reimplementation in mind. Because the original algorithm is an integral part of ROSETTA, and depended upon by other parts of the system, it could not be altered in any way to allow easy integration of the new representation. Thus, parts of the algorithm had to be re-implemented to be able to use the new representation.

A.3.3 AUC Computation

As briefly mentioned in the previous section, performance optimization was an important issue in the implementation. The genetic algorithm is very computation intensive; a typical run of the rule filtering algorithm can take several hours. Particularly, the performance evaluation in the fitness function is a bottleneck; consisting of constructing ROC curves from (typically) several hundred objects several hundreds times for each of (typically) several thousands of generations. The implemented AUC calculation routine is shown in Figure A.2. This routine uses

```

function FIND-FIRING(TestSet, RUL) returns fires, a list of rule sets
  inputs:
    TestSet, a decision system
    RUL, the unfiltered rule set

  for all  $x \in \textit{TestSet}$  do
     $\textit{fires}_x \leftarrow \text{MATCHES}(\textit{RUL}, x)$ 
  end for
  return fires
end function

```

Figure A.1: Construction of the fires table

the table *fires*. This table is constructed at the creation of the `fitness` object; as shown in Figure A.1. The *fires* table maps an object in the decision system used to measure the AUC (the hold-out set) into the set of rules from the unfiltered rule set that have matching antecedents. These rule sets only need to be constructed once for a GA run. When calculating the certainty $\phi(x)$ given by a rule set *RUL'* for an object *x*, the intersection between *RUL'* and the set of firing rules for *x* is used to classify the object according to the classification scheme outlined in Section 3.6. Together with the decision value $d(x)$ collected from *x*'s decision system, $\phi(x)$ for all objects *x* constitutes a list of pairs used to construct the ROC curve, and the AUC using the trapezoidal rule. The functions `MATCHES`, `CREATEROCCURVE` and `CALCULATEAUC` are available in the ROSETTA library.

In the actual implementation of `EVALUATE-PERFORMANCE`, rule sets (individuals) are represented by integer vectors. In addition to generally being a much more suitable representation than that in the original algorithm, this representation renders possible efficient calculation of the intersection of represented rule set and the list of firing rules for an object.

```

function EVALUATE-PERFORMANCE(RUL, TestSet) returns auc, an AUC value
  inputs:
    RUL, a rule set
    TestSet, a decision system

  for all  $x \in \text{TestSet}$  do
     $RUL' \leftarrow RUL \cap \text{fires}_x$ 
     $\phi(x) \leftarrow \text{CLASSIFY}(RUL', x)$ 
     $\text{pairs} \leftarrow \text{pairs} \cup (\phi(x), d(x))$ 
  end for
   $\text{roc} \leftarrow \text{CREATEROCCURVE}(\text{pairs})$ 
   $\text{auc}_l \leftarrow \text{CALCULATEAUC}(\text{roc})$ 
  return auc
end function

```

Figure A.2: Performance evaluation of a rule set

A.3.4 User Interface

The user interface to the algorithm, implemented in the ROSETTA front-end, is shown in Figures A.3 and A.4.

A.4 The Quality Based Algorithm

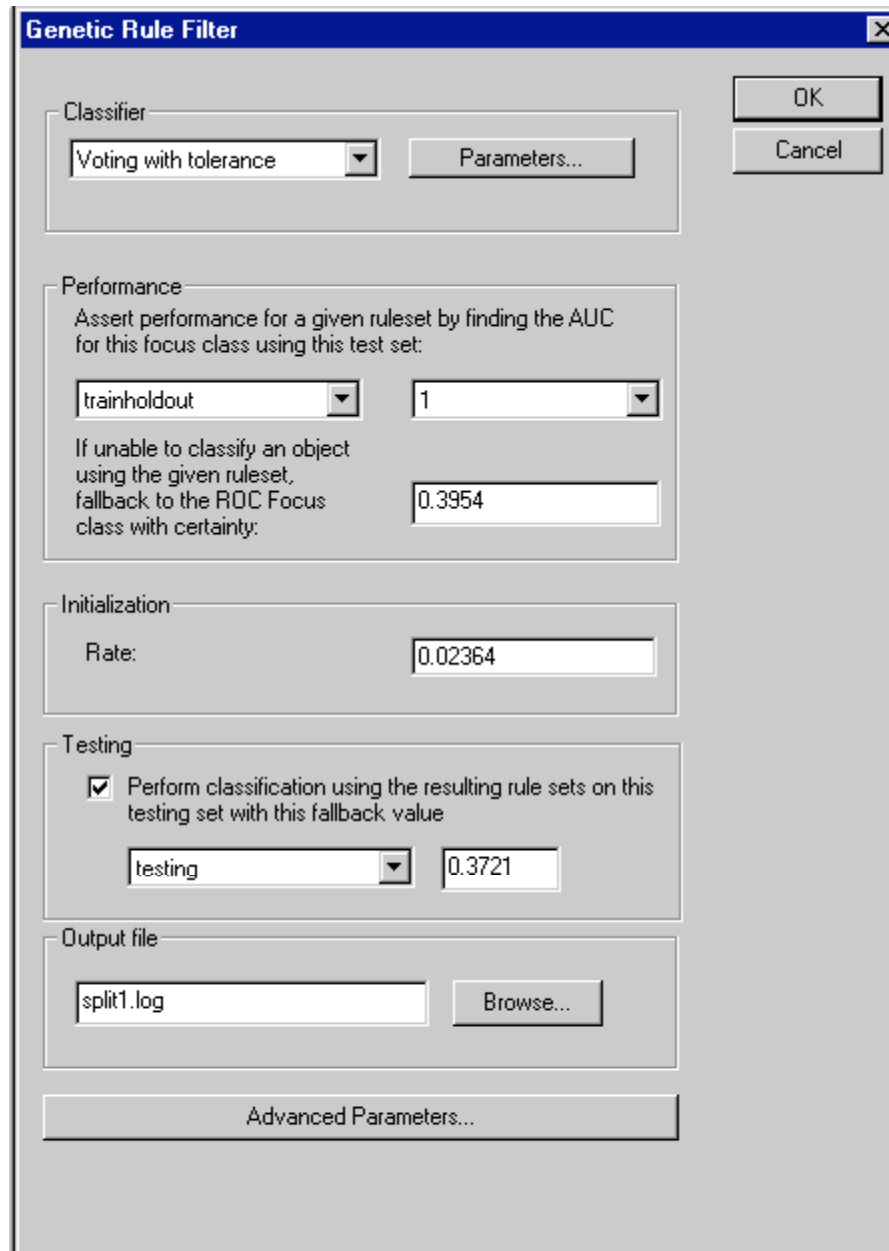
The quality based rule filtering method is implemented as two algorithm objects. The first is the implementation of the graph construction algorithm; shown in Figure 7.1. Similar to the genetic filtering algorithm, this algorithm takes a rule set as input and produces a log file as a side effect. The log file contains the AUC versus threshold graph. The algorithm is implemented by iterating over an associative map giving the corresponding rule set for each quality value, going from high quality values towards lower. The algorithm can be interrupted when all the sufficiently small rule sets have been evaluated; it is not necessary to create the whole graph. At the initialization of the algorithm, a *fires* table is constructed, and the AUC is calculated the same way as in the genetic algorithm (Figure A.2) for each generation. The second algorithm object is the implementation of the actual filtering algorithm, shown in Figure 7.2. This algorithm takes a rule set as input, and returns the filtered rule set. The quality measures are calculated from the rule sets' originating decision systems.

A.4.1 A Note on Rule Representation

Some of the learning algorithms in ROSETTA in some cases generate rules on the form:

$$\alpha \rightarrow \beta_1 \vee \beta_2 \tag{A.1}$$

Each of the disjuncts β_i in the consequent is accompanied by a support count $\text{support}(r_i)$ (relative to the rule's originating decision system), corresponding to the rule $r_i = \alpha \rightarrow \beta_i$. For classification purposes one such rule can be interpreted as a set of rules: $\{\alpha \rightarrow \beta_1, \alpha \rightarrow \beta_2\}$. Generally, then, the quality estimate of a rule



The image shows a software dialog box titled "Genetic Rule Filter". It contains several sections for configuring a genetic algorithm. The "Classifier" section has a dropdown menu set to "Voting with tolerance" and a "Parameters..." button. The "Performance" section includes a dropdown for "trainholdout" set to "1", a text input for a fallback value of "0.3954", and explanatory text. The "Initialization" section has a "Rate:" label and a text input with "0.02364". The "Testing" section has a checked checkbox for performing classification on a testing set, a dropdown set to "testing", and a text input with "0.3721". The "Output file" section has a text input with "split1.log" and a "Browse..." button. At the bottom is an "Advanced Parameters..." button. "OK" and "Cancel" buttons are in the top right corner.

Genetic Rule Filter

Classifier
Voting with tolerance Parameters...

Performance
Assert performance for a given ruleset by finding the AUC for this focus class using this test set:
trainholdout 1
If unable to classify an object using the given ruleset, fallback to the ROC Focus class with certainty: 0.3954

Initialization
Rate: 0.02364

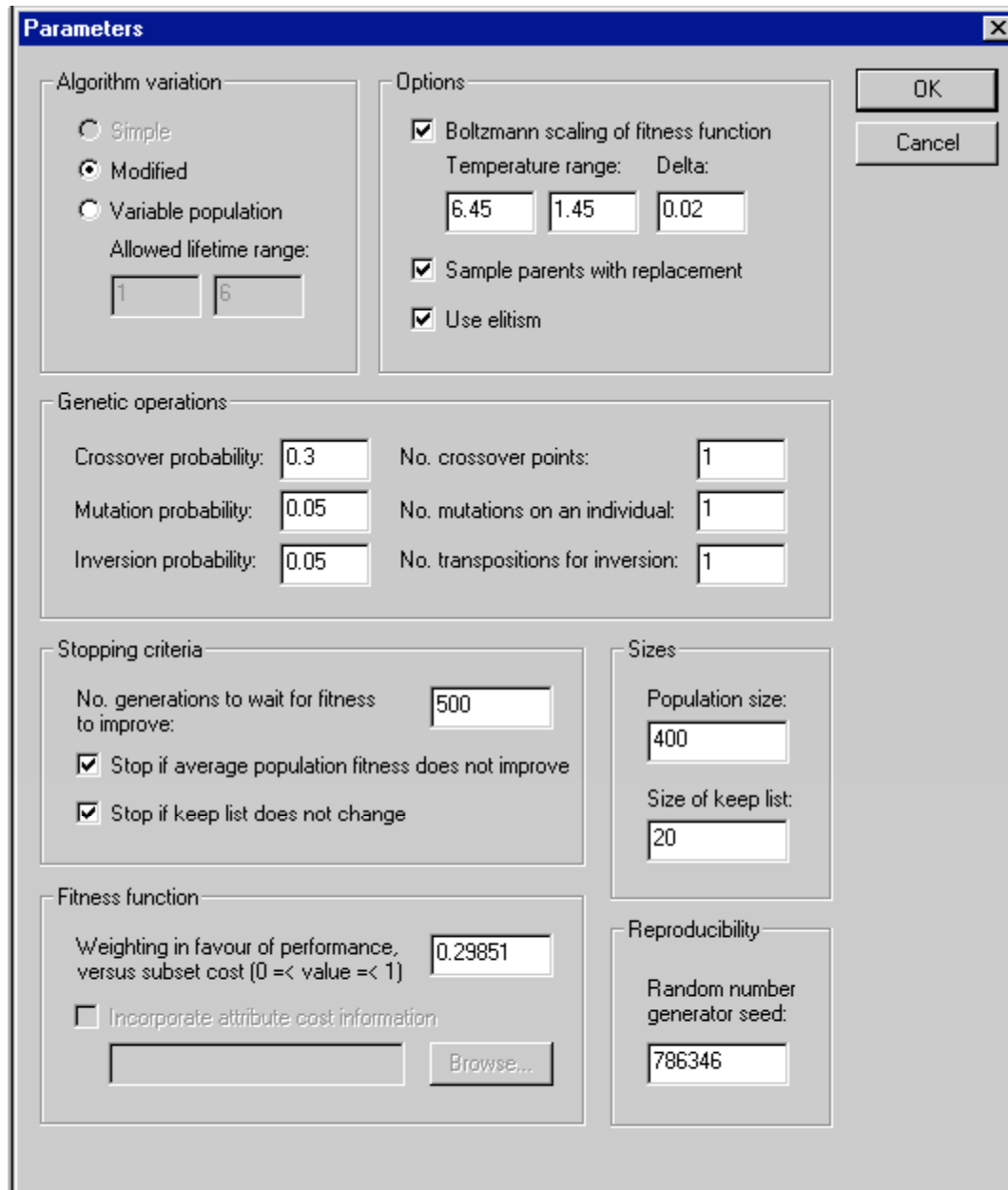
Testing
☒ Perform classification using the resulting rule sets on this testing set with this fallback value
testing 0.3721

Output file
split1.log Browse...

Advanced Parameters...

OK
Cancel

Figure A.3: User interface to the genetic algorithm



The image shows a 'Parameters' dialog box for a genetic algorithm. It is divided into several sections: 'Algorithm variation' with radio buttons for 'Simple', 'Modified' (selected), and 'Variable population', and a 'Allowed lifetime range' of 1 to 6; 'Options' with checkboxes for 'Boltzmann scaling of fitness function' (checked), 'Sample parents with replacement' (checked), and 'Use elitism' (checked), along with 'Temperature range' (6.45 to 1.45) and 'Delta' (0.02); 'Genetic operations' with 'Crossover probability' (0.3), 'Mutation probability' (0.05), 'Inversion probability' (0.05), and counts for crossover points (1), mutations (1), and transpositions (1); 'Stopping criteria' with 'No. generations to wait for fitness to improve' (500) and checkboxes for 'Stop if average population fitness does not improve' and 'Stop if keep list does not change'; 'Fitness function' with 'Weighting in favour of performance, versus subset cost (0 =< value =< 1)' (0.29851) and an option to 'Incorporate attribute cost information' with a 'Browse...' button; 'Sizes' with 'Population size' (400) and 'Size of keep list' (20); and 'Reproducibility' with a 'Random number generator seed' (786346). 'OK' and 'Cancel' buttons are in the top right.

Parameters

Algorithm variation

☐ Simple
☒ Modified
☐ Variable population

Allowed lifetime range:
1 6

Options

☒ Boltzmann scaling of fitness function
Temperature range: 6.45 1.45 Delta: 0.02
☒ Sample parents with replacement
☒ Use elitism

Genetic operations

Crossover probability: 0.3 No. crossover points: 1
Mutation probability: 0.05 No. mutations on an individual: 1
Inversion probability: 0.05 No. transpositions for inversion: 1

Stopping criteria

No. generations to wait for fitness to improve: 500
☒ Stop if average population fitness does not improve
☒ Stop if keep list does not change

Fitness function

Weighting in favour of performance, versus subset cost (0 =< value =< 1) 0.29851
☐ Incorporate attribute cost information
Browse...

Sizes

Population size: 400
Size of keep list: 20

Reproducibility

Random number generator seed: 786346

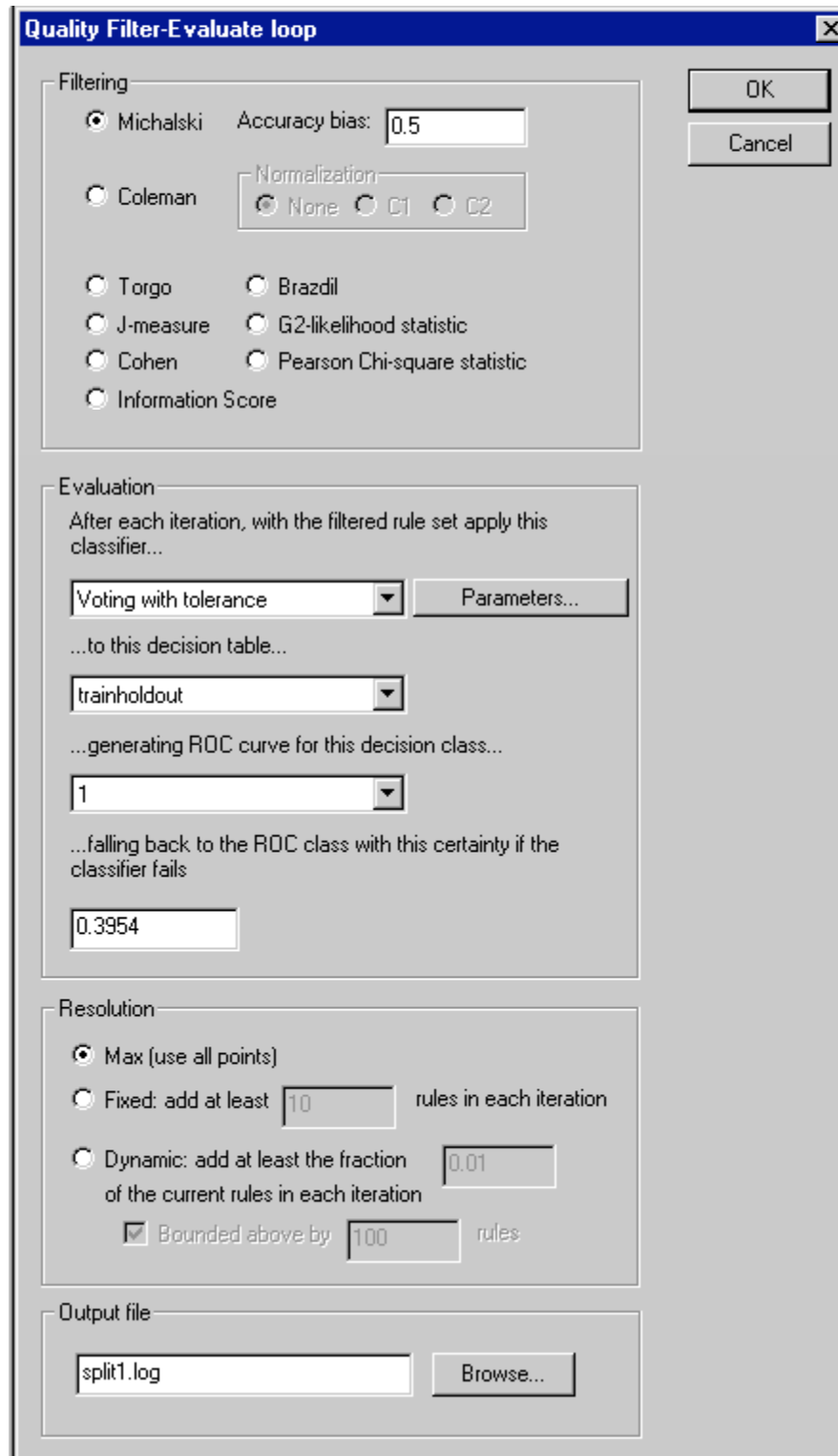
OK Cancel

Figure A.4: Advanced parameters for the genetic algorithm

on the form in Equation A.1 can be defined as equal to the *best* quality estimate of the rules in this set. The association based quality formulas, however, are only valid for rules $\alpha \rightarrow \beta$ where β denotes the dominating decision class, as discussed in Section 7.2.2. By using the “best quality” calculation above, these formulas will be invalidated if the disjunct with the lowest support count is selected for quality calculation. Thus, in order to prevent invalidation of the formulas, the rule $\alpha \rightarrow \beta_i$ with the highest support count is used to estimate association based quality for rules on the form given in Equation A.1, while the corresponding rule with best quality value is used for the rest of the formulas.

A.4.2 User Interface

The user interfaces to the algorithms, implemented in the ROSETTA front-end, are shown in Figures A.5 and A.6.



The dialog box is titled "Quality Filter-Evaluate loop" and contains four main sections: Filtering, Evaluation, Resolution, and Output file.

Filtering

- ☒ Michalski Accuracy bias:
- ☐ Coleman Normalization:
 - ☒ None
 - ☐ C1
 - ☐ C2
- ☐ Torgo
- ☐ Brazdil
- ☐ J-measure
- ☐ G2-likelihood statistic
- ☐ Cohen
- ☐ Pearson Chi-square statistic
- ☐ Information Score

Evaluation

After each iteration, with the filtered rule set apply this classifier...

Parameters...

...to this decision table...

...generating ROC curve for this decision class...

...falling back to the ROC class with this certainty if the classifier fails

Resolution

- ☒ Max (use all points)
- ☐ Fixed: add at least rules in each iteration
- ☐ Dynamic: add at least the fraction of the current rules in each iteration
 - ☒ Bounded above by rules

Output file

Browse...

OK Cancel

Figure A.5: User interface to the quality based graph algorithm

Quality-based Rule Filter [X]

Quality Measure

☒ Michalski Accuracy bias:

☐ Coleman

Normalization

☒ None ☐ C1 ☐ C2

☐ Torgo ☐ Brazdil

☐ J-measure ☐ G2-likelihood statistic

☐ Cohen ☐ Pearson Chi-square statistic

☐ Information Score

Remove

☒ Rules with quality lower than

☐ Rules with quality higher than

OK Cancel

Figure A.6: User interface to the quality based filtering algorithm

APPENDIX B

Quality Distributions

B.1 Acute Appendicitis Data

The distributions of the quality values for the three rule sets in the acute appendicitis experiment are shown in Figure B.1. The number of rules with undefined quality for the J-measure is not shown.

B.2 Cleveland Data

The distributions of the quality values for the three rule sets in the Cleveland experiment are shown in Figure B.2. The number of rules with undefined quality for the J-measure is not shown.

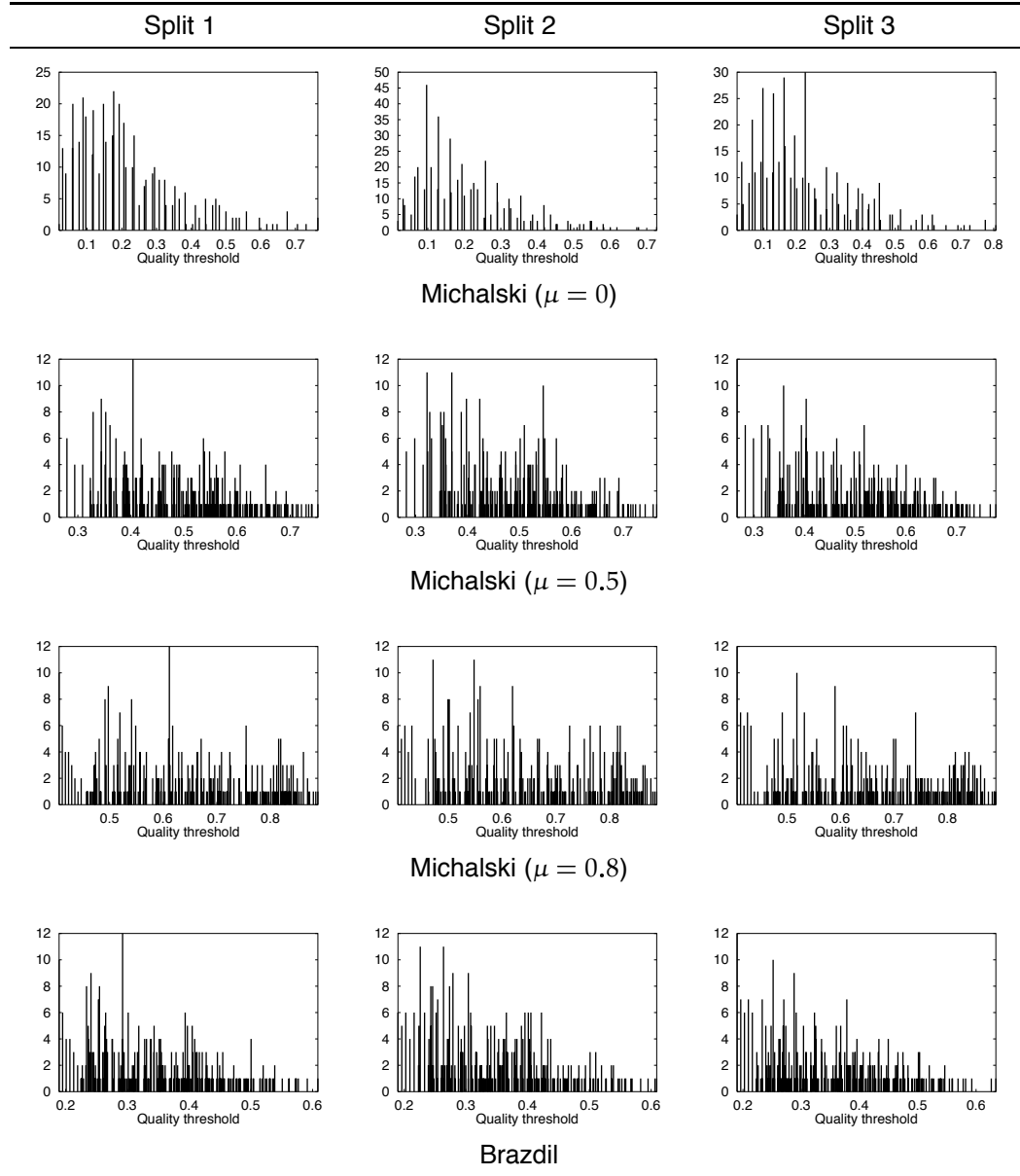


Figure B.1: Quality value distributions. (Continues on page 139)

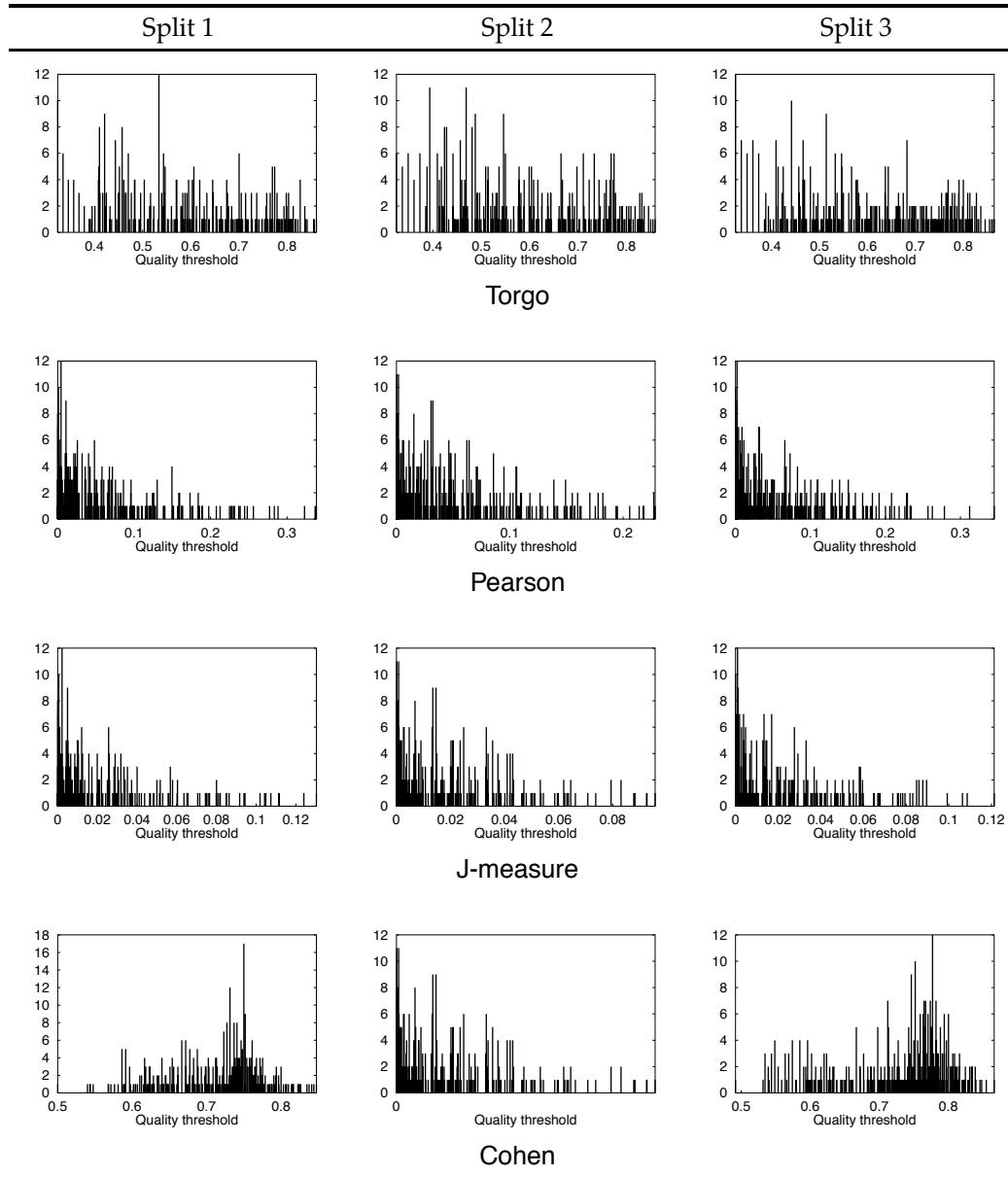


Figure B.1: (Continued, continues on page 140)

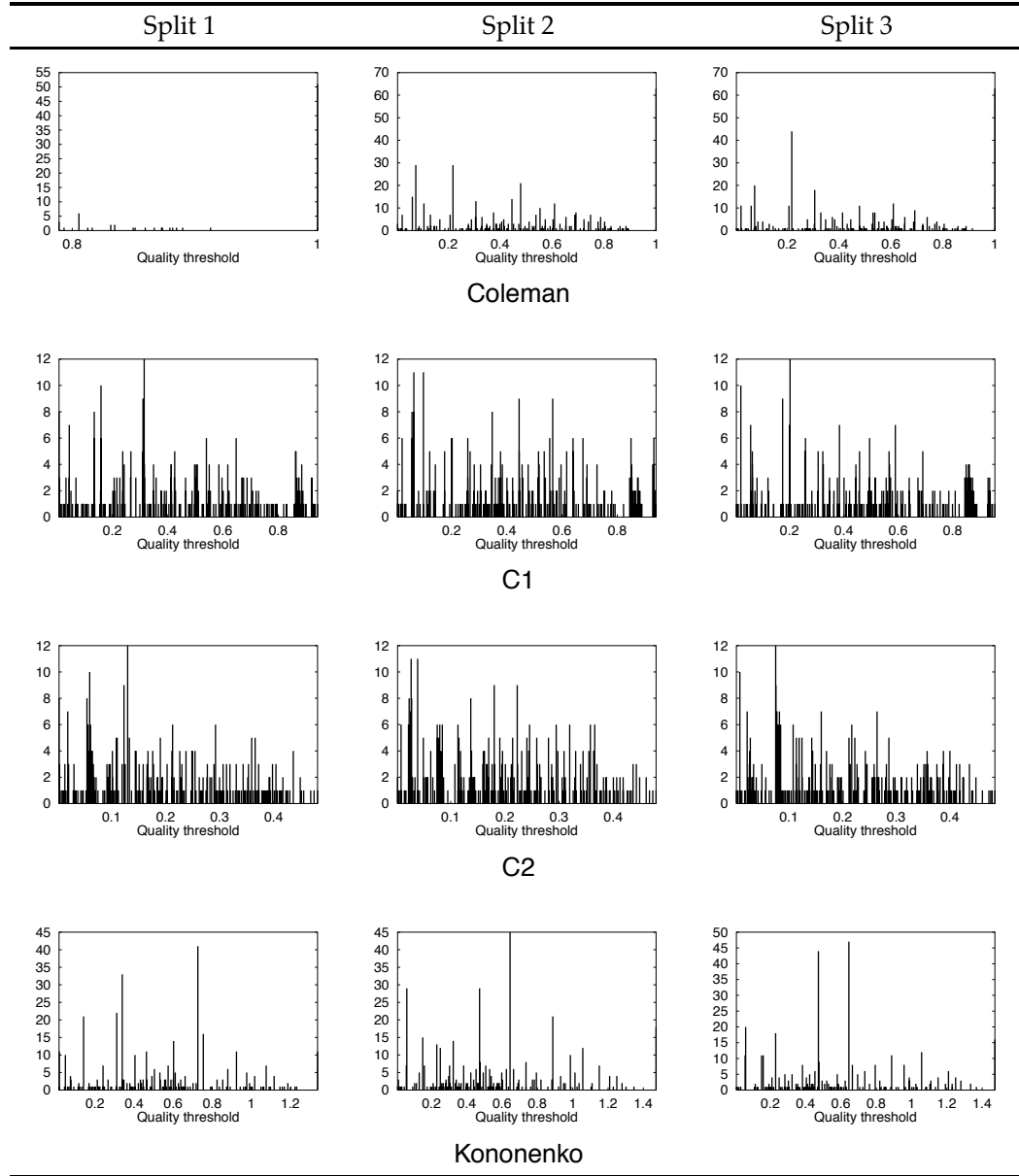


Figure B.1: (Continued)

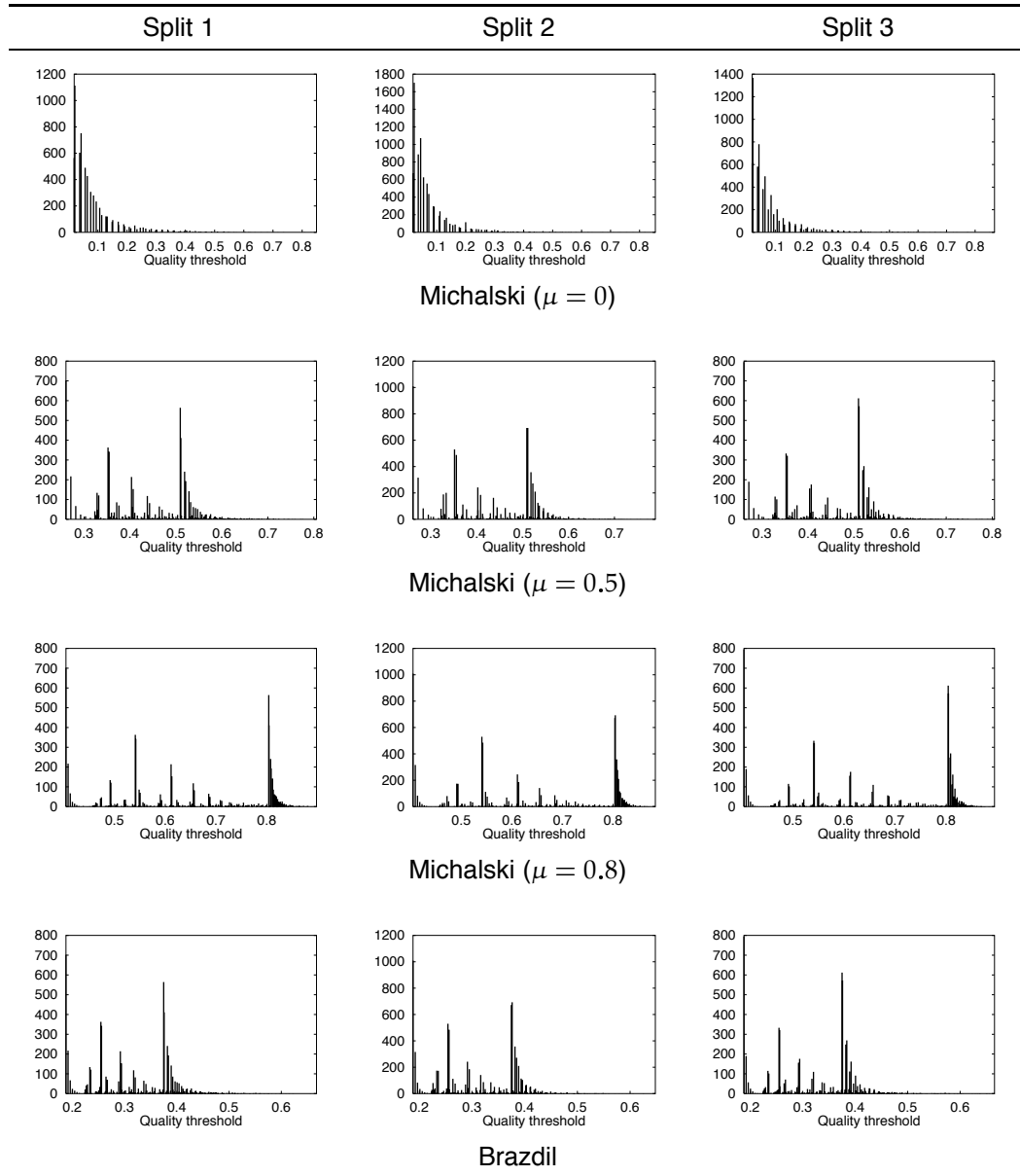


Figure B.2: Quality value distributions. (Continues on page 142)

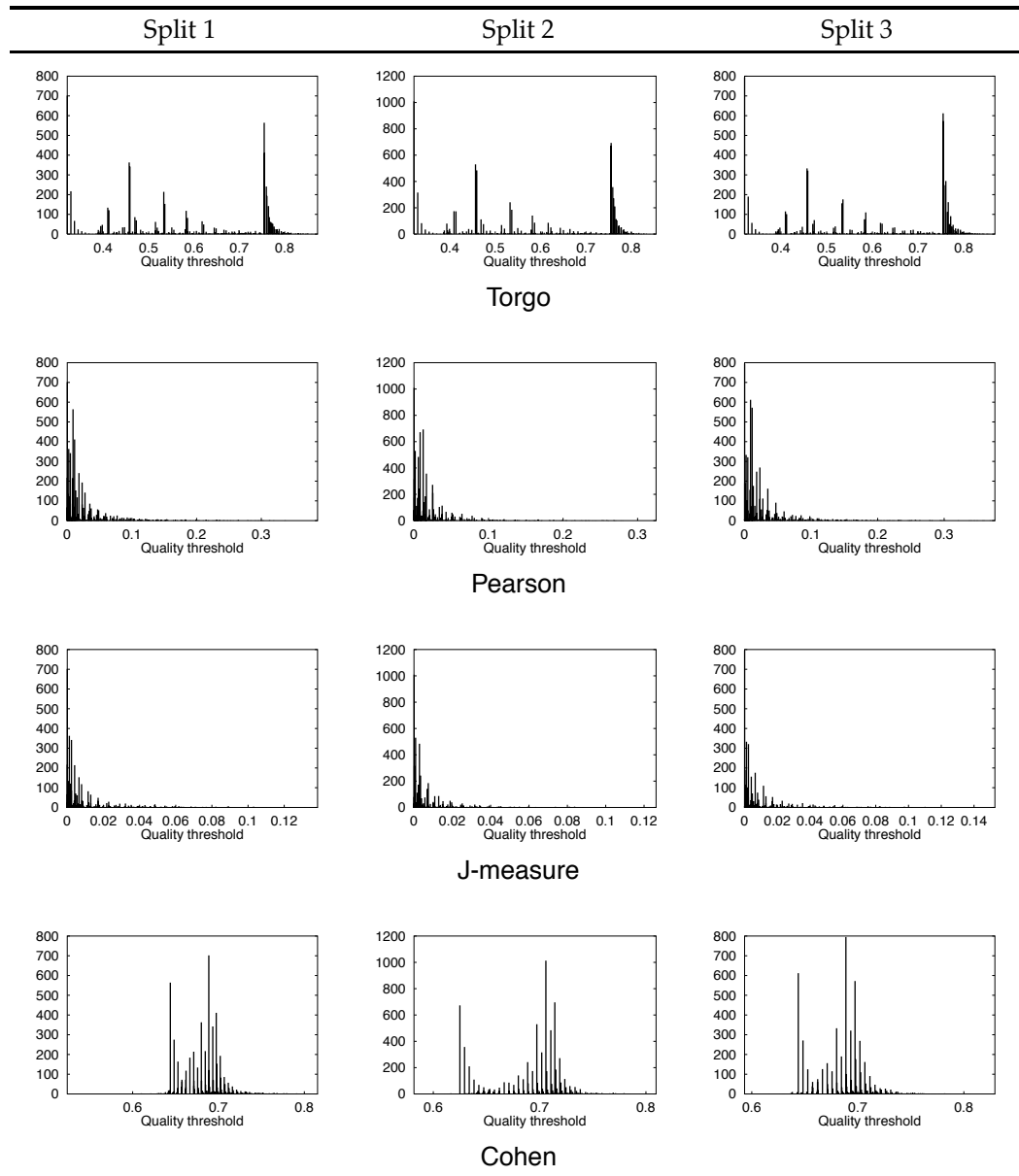


Figure B.2: (Continued, continues on page 143)

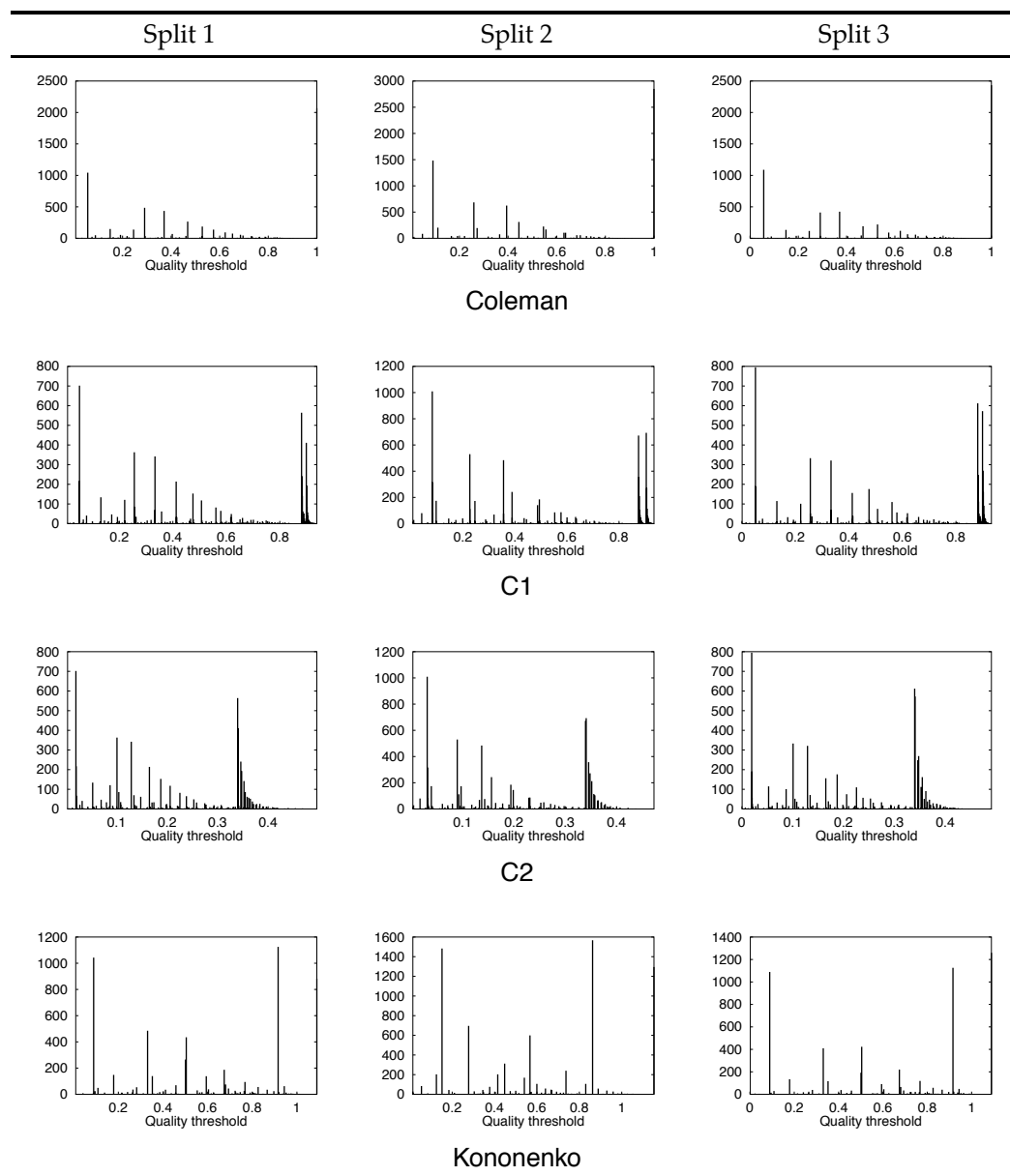


Figure B.2: (Continued)