META MINING SYSTEM FOR SUPERVISED LEARNING

by

LUKASZ KURGAN

B.S., AGH University of Science and Technology, Krakow, Poland, 1998

M.S., AGH University of Science and Technology, Krakow, Poland, 1999

A thesis submitted to the

Faculty of the Graduate School in partial fulfillment

of the requirement for the degree of Doctor of Philosophy

Department of Computer Science

2003

this thesis entitled:

Meta Mining Architecture for Supervised Learning

written by Lukasz Kurgan

has been approved for the Department of Computer Science

Dr. Krzysztof Cios (Chair of the Committee)

Dr. James Martin (Committee Member)

Date

a final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Kurgan, Lukasz (Ph.D., Computer Science)

Meta Mining System for Supervised Learning

Dissertation directed by Professor Krzysztof Cios

Supervised inductive machine learning is one of several powerful methodologies that can be used for performing a Data Mining task. Data Mining aims to find previously unknown, implicit patterns that exist in large data sets, but are hidden among large quantities of data. These patterns describe potentially valuable knowledge. Data Mining techniques have been focused on finding knowledge, often expressed in terms of rules, directly from data. More recently, a new Data Mining concept, called Meta Mining, was introduced. It generates knowledge utilizing two-step procedure, where first meta-data is generated from the input data, and next the meta-data is used to generate meta-rules that constitute final data model.

In this dissertation we examine a new approach to generation of knowledge, using supervised inductive learning methodologies combined with Meta Mining. We propose a novel data mining system, called MetaSqueezer, for extraction of useful patterns that carry new information about input supervised data set. The major contribution of this thesis is design and development of the above system, supported by extensive benchmarking evaluation results. Two key advantages of the system are its scalability, which results from its linear complexity, and high compactness of user-friendly data models that it generates. These two features make it applicable for applications that use megabytes, or even gigabytes of data.

The fields contributing to this research are Inductive Machine Learning, Data Mining and Knowledge Discovery, and Meta Mining. A study of existing Machine Learning methodologies, which give similar results, is given to properly situate the research and to help in evaluation of the system.

The usefulness of the system is evaluated theoretically and also empirically via thorough testing. The results show that the system generates very compact data models. They also confirm linear complexity of the system, which makes it highly applicable to real data.

Results of application of the system to cystic fibrosis data are provided. This application generated very useful results, as evaluated by the domain experts.

Dedication

To my brother, Michal, with the best wishes for successful and self-fulfilling life-long journey.

Acknowledgements

I would like to thank all who helped me to complete this dissertation.

First, my appreciation goes to my advisor, Dr. Krzysztof Cios, whose experience and continuous support helped me over the last few years to become a successful researcher. I especially thank him for understanding and encouragement for my research directions, which resulted in establishing a fruitful and strong advisor-student relationship. You are the best!

I am grateful to the members of my dissertation committee, Dr. Andrzej Ehrenfeucht, Dr. Clayton Lewis, Dr. Dennis Lezotte, and Dr. James Martin. Their very much appreciated input helped me to clarify my understanding of the subject, and thus to improve the quality of this work. I would like to thank Dr. Frank Accurso and Marci Sontag, for providing cystic fibrosis data and helping to understand the problems to be modeled. They not only helped me to provide a strong validation for the system that is described here, but also enabled me to feel importance of my research, by showing its applicability to important medical problems. I also would like to thank Dr. Tom Altman for his valuable comments and correction that greatly helped in improving and polishing this dissertation.

Last, but not least, I would like to thank my family for their continuing support. I thank my parents, who always tried to explain that knowledge is the key to the success, and for their continuing words of encouragement.

Contents

Tables

Figures

Chapter 1

1 Introduction

This chapter provides an overview of Inductive Learning, with emphasis on Data Mining and Machine Learning. It also explains the motivation and goal of the research and talks about potential applications areas.

1.1 Overview of Inductive Learning

Humans attempt to understand their environment by using its simplified version, called a model. Creation of such a model is called Inductive Learning (IL). During the learning phase, humans try to recognize similarities among objects and events in the environment, by observation. Next, they group similar objects into classes and construct hypotheses used to predict behavior of the members (examples) from these classes (Holland et al., 1986). The same activities are also performed when knowledge is generated from input data. The generated knowledge is not a mere copy of the input data, but rather consists of information that is inferred from the data. Two inference techniques, which are used to derive new knowledge, are (Holsheimer and Siebes, 1994):

- deduction, which is a technique that infers knowledge that is a logical consequence of the information in the input data. It can be used if the data describing some domain is proven to be correct. Most database management systems (DBMSs) (e.g., relational DBMSs) can perform deduction via use of simple operators, like the join operator. Such join operator, when applied to two relational tables, infers a relation between them. A research area called deductive databases uses the idea of deduction to provide a user with answers to queries (Ullman, 1989; Ullman and Zaniolo, 1990). Such database is augmented with an inference system that is used to derive rules that are used to answer user's queries.

- induction, which infers knowledge that is generalized from the information in the input data. The induction works by searching for regularities among the data to provide high level summary of the information contained in the data. It is usually performed in the form of a search for a correct hypothesis (rule), or a set of them, which is guided by the examples from the input data. IL generates hypotheses that are always correct for the input data, but only plausible outside of the data.

1.1.1 Basic Definitions

Information about the environment is represented by a training data set. Since most of current DBMSs are relational, the most popular representation form for the data is a relational table, which consists of tuples. Each tuple represents one or more objects (examples). We assume that the table stores information about properties of the examples, in terms of attributes, but not the relationship between the examples. We also assume that the training set consist of a single table. The second assumption can be easily realized for all database schemas and queries (Ullman, 1989). The table may include unknown values, which means that some of the attributes describing examples may be not known during the learning process.

Definition 1. *Training Set.*

Let $A = \{A_1, A_2, ..., A_k\}$ be a set of attributes with their domains Dom_1, Dom_2, …, Dom_k. The training set is a finite subset of universe $U = Dom_1$ x Dom_2 x … x Dom_k, and is defined as a table over A. An example is a tuple in the training set.

The user of training set defines classes, also called concepts, in the data (Michalski, 1983a). We assume that the training set contains one or more attributes, called predicted attributes, which are used to denote the class of an example. The remaining attributes are called predicting attributes. A class is defined by a condition that involves combination of values of the predicted attributes.

Definition 2. *Class.*

A class C_i is a subset of the training data set S, consisting of all objects that satisfy the class condition $cond_i$: $C_i = \{\text{examples} \in S \mid cond_i(o)\}$. Examples that satisfy $cond_i$ are called positive examples, or examples of class C_i. The remaining examples are called negative examples.

When the classes are defined, an IL algorithm can infer rules, or class descriptions, from the training set. The rules are generated for each of the classes, using only the predictive attributes. Rules generated for class C_i should describe only the positive, and none of the negative examples.

Definition 3. *Rule.*

Let A be the set of predicting attributes. A rule that describes class C_i is a formula IF $(A_1 = c_1) \wedge (A_2 = c_2) \wedge \ldots \wedge (A_n = c_n)$ THEN C_i, such that:

1. $A_i \in A$ and $i \neq j \rightarrow A_i \neq A_j$
2. $c_i \in Dom(A_i)$

A rule can be also written as IF D THEN C, where D is called description and C denotes the class for the rules was generated. The description is a disjunction of elementary statements (i.e., $(A_i = c_i)$ called selectors. A rule generated for class C_i is correct with respect to the training set if its description covers positive examples, and none of negative examples.

The process of rule generation performed by an IL algorithm is shown in Figure 1.

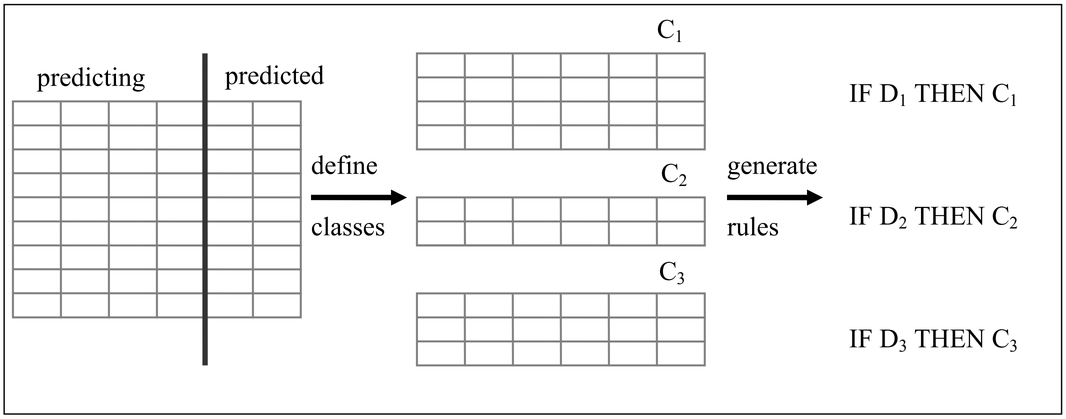


Figure 1. Process of learning rules by an IL algorithm

1.1.2 Supervised Inductive Learning

Two types of IL techniques can be distinguished (Carbonell et al, 1983). In supervised learning, known also as learning from examples, an external teacher defines classes and provides examples for each class. The learning algorithm infers rules that describe common properties exhibited by the examples from the training data for each class. These class descriptions usually take form of classification rules "IF <description> THEN <class>", which can be used to predict the class of previously unseen examples or to find new and useful regularities exhibited in the data. The teacher can define either one or multiple classes (Dietterich and Michalski, 1983). In case of the single-class learning, the teacher defines a single class C for which an IL algorithm generates rules. The description must distinguish examples from class C (positive examples) from examples which do not belong to class C (negative examples). This type of learning is also called binary learning, since it used two classes of examples: positive and negative. In multiple-class learning, the teacher defines a finite number of classes C_1, C_2, …, C_c, for which an IL algorithm must find descriptions. The algorithm generates rules for the ith class by treating examples from the class C_i as positive examples, and examples belonging to any other class as negative examples.

In unsupervised learning, also known as learning from observation, inductive algorithm discovers the classes by itself based on common properties of examples. Detailed discussions of these two learning models can be found in

(Holsheimer and Siebes, 1994). In this work we are only concerned with the development of a supervised IL system that performs both single- and multiple-class learning tasks from supervised training data.

1.1.3 Inductive Learning via Search

The goal for any supervised IL algorithm is to generate a set of correct rules for each of the classes defined by the teacher. The simplest approach to generate such rules is to perform exhaustive search, also known as learning by enumeration, which tries all possible descriptions to find these that best fit the class that is currently being described. There are two major problems when learning by enumeration. First, it is extremely complex, which leads to a very long learning time. Second, the rules generated by enumeration will describe only examples from some training set, when they should provide generalized description of the examples that will also be able to describe examples from outside of the learning set. Before we describe other learning methods, which are designed to overcome the above problems, we provide several necessary definitions.

1.1.3.1 The Search Space

The search space $<D, O, f>$ consists of a set of descriptions D, a set of operations on these descriptions O, and a quality function f (Holsheimer and Siebes, 1994).

The description space *D* is a set of all possible descriptions for a particular training set. Each description D has its corresponding subset of the training set S, called cover $\sigma_D(S)$.

Two main operations on *D* are: generalization and specialization. The goal of the generalization is to weaken the description by making it to cover more examples. On the other hand, specialization strengthens the description by reducing the coverage.

Let assume that generalization operation was used to convert description *D* into description *D'*. If an example is covered by *D*, it is also covered by *D'*, but the reverse does not hold. Thus, the generalization operation is not truth preserving. Simply, if *D* is correct, its generalization *D'* may not be correct.

Definition 4. *Generalization.*

A set-description $D = (A_1 \in S_1 \wedge \ldots \wedge A_i \in S_i \wedge \ldots \wedge A_n \in S_n)$ is generalized by extending the set of values for a particular attribute A_i to S_i' where $S_i \subset S_i' \subseteq Dom_i$.

Specialization is an inverse of the generalization operation, where S_i is replaced with S_i', and $S_i \supset S_i'$.

Quality function f is used to indicate quality of particular descriptions. Each description must satisfy two conditions. First, it must be valid, which means that it must classify unseen, i.e., not present in training set, examples correctly. Second, it should be correct in terms of providing correct description with respect to one of the classes defined by the user.

Validity of the generated rules can be assessed using two criteria. First, after the rules are generated, a classification test is performed. Such test applies generated rules to test set, which was unseen during the learning process. The rules are used to classify examples from the test set, and the results are compared with class values, which are assigned to the test examples by the teacher. Accuracy, defined below, is computed, and higher validity value is assigned to rules that achieve higher accuracy. Second, the Ockham's razor principle is used. It says that the simpler the description, the more likely it is that it correctly describes existing pattern in the data. The complexity of descriptions is most commonly evaluated by the number of selectors that the description uses. Thus, the validity is higher for simpler descriptions, or smaller complexity.

Correctness of the generated rules is assessed by stating that the entire set of descriptions D for class C_i should cover all positive examples, and none of negative examples, i.e., $\sigma_D(S) = C_i$. Most IL algorithms relax this requirement by requiring that the description should describe significant majority of positive examples, and may cover small number of negative examples. This relaxation is due to data inconsistencies, i.e., the same example describes different classes, missing values, and noise. It is usually assumed that correctness of the rules is verified based on the design of IL algorithm that generates them.

1.1.3.2 Rule Validity Tests

The most popular test to verify validity of generated rules is the accuracy test. The test is defined as probability that an example covered by a description actually belongs to the class that is described by that description.

$$accuracy = \frac{TP}{total}100\%$$

where *TP* (true positive) is the number of correctly recognized test examples, and *total* is the total number of test examples. A test example is checked against all rules describing all classes.

Also, a more precise test measure called verification test can be used to test the rules. The verification test is a standard test procedure in medicine, where sensitivity and specificity analysis is used to better evaluate confidence in the results (Cios and Moore, 2002). This work applies both, the accuracy and verification tests, to provide a thorough verification of validity of rules generated by the MetaSqueezer system.

The verification test consists of three evaluation criteria:

$$sensitivity = \frac{TP}{hypothesis \quad positive}100\% = \frac{TP}{TP + FN}100\%$$

$$specificity = \frac{TN}{hypothesis \quad negative}100\% = \frac{TN}{FP + TN}100\%$$

$$predictive \quad accuracy = \frac{TP + TN}{total}100\% = \frac{TP + TN}{TP + TN + FP + FN}100\%$$

where *TP* (true positive) is the number of correct positive classifications, *TN* (true negative) is the number of correct negative predictions, *FP* (false positive) is the number of incorrect positive predictions, and *FN* (false negative) is the number of incorrect negative predictions. These values are computed based on the possible test outcomes shown in Table 1.

Table 1. Outcomes of the verification test.

	test result positive	test result negative
positive rule	TP	FN
negative rule	FP	TN

The predicative accuracy is equivalent to the accuracy. The remaining two criteria give additional insight into the goodness of the generated rules. The sensitivity measures how many of the examples described by the rules as positive were truly positive. The specificity measures how many of the examples described by the rules as negative were truly negative. Specificity and sensitivity enable evaluation of how the rules perform on the positive and negative data separately. This is very important when the numbers of positive and negative examples are different. In this case, the accuracy provides just the average result for positive and negative examples together, when true accuracy for positive examples can be very different than accuracy for the negative examples. The difference can be easily noticed when using sensitivity and specificity. Only the results with high values for all three criteria can assure high confidence level in the rules. A study by Kukar (Kukar et al., 1999) shows the importance of using, and trade-offs between, the sensitivity and specificity. Advantages of using verification test versus accuracy test were also shown in (Cios et al., 1998).

1.1.3.3 Search Algorithms

IL can be seen as a search over the defined search space. The general approach is to start with an initial description, and iteratively modify it via one of the operators until its quality exceeds a user-defined threshold (Holsheimer and Siebes, 1994). Two approaches for generation of rules are: data-driven, or bottom-up approach, and model-driven, or top-down approach.

In the bottom-up search the initial description consists of all examples from the training set that describe positive class. The initial description is correct with respect to training set, but is too complex, and thus during search it is modified by applying generalization operator. The search is based on multiple generalizations, which are applied until the description is correct, with a certain tolerance, with respect to the training set. The bottom-up search is shown in Figure 2a, where filled circles depict positive examples, empty circles the negative examples, and the description is shown by the shaded areas. The bigger the shaded area, the more general the description is.

In the top-down search the initial description describes the entire universe, or its subset, of the training set. The initial description is refined by applying both generalization and specialization operators, until the quality of the description exceeds a threshold. The top-down search is shown in Figure 2b.

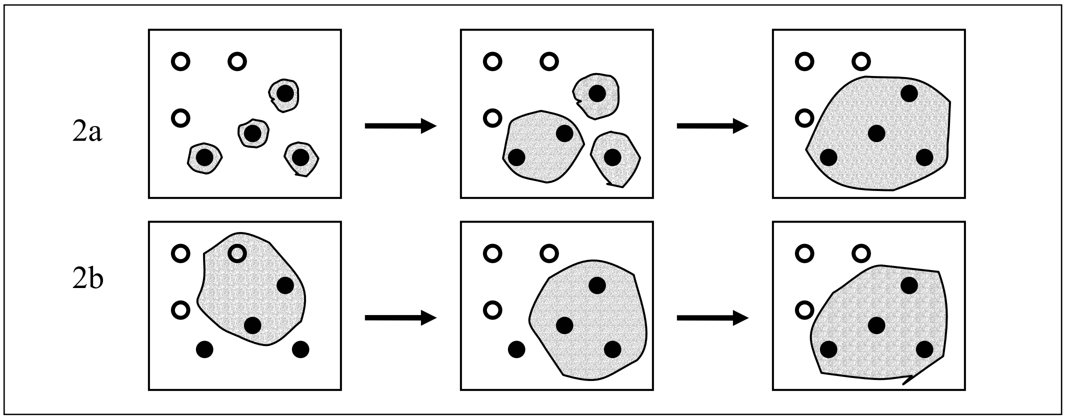


Figure 2. Rule search process: a) bottom-up approach, b) top-down approach

The search space can be modeled by a directed graph, where nodes represent descriptions, and arcs represent operations. Figure 3 shows a sample search graph, where a set of operations in *O* is applied to the initial description D_1 to generate new descriptions D_{2i}. The process of application of operations is repeated until a goal node, which represents a description of sufficient quality, is reached. Thus, the graph can be used to represent a search for rules, where a particular sequence of operations is used to generate correct rules.

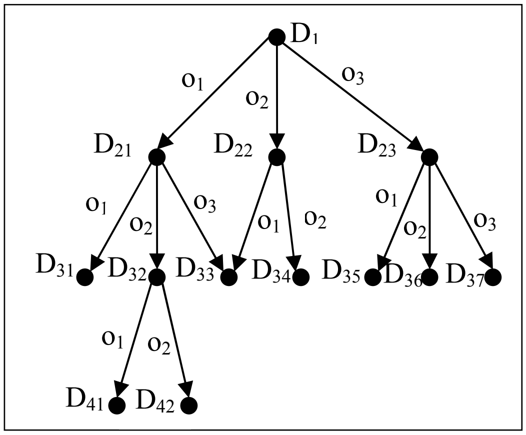


Figure 3. Sample search graph

Two search strategies are: the *irrevocable* search, and the *tentative* search (Nillson, 1982). In the irrevocable search, once an operation is selected and applied to a description, it will never be reconsidered. This search executes a single path thought the graph to generate a rule. In the tentative search, the once an operation is selected and applied to a description, it may still be undone, to perform a different operation for that description. The main reason for undoing an operation is that the search identified that selection of a different operation would lead to a better rule. The disadvantage of the latter search strategy is its high computational cost, when compared with the irrevocable search.

Both of these search strategies can work in uninformed or informed mode. In uninformed mode, an operation is chosen arbitrarily. In this case, using irrevocable strategy leads to performing a depth-first search, while using tentative strategy leads to either depth-first search with backtracking, or breath-first search (Russel and Norvig, 1995). Uninformed searches are computationally expensive, since many descriptions are generated, and for each of them correctness must be evaluated. Informed, or heuristic, searches select operations based on a predefined heuristic. The heuristic is used to select descriptions which are on the shortest path to a goal node, and thus reducing computational complexity.

Finally, there are three types of search strategies: exhaustive search, beam search, and hill climber search. The simplest strategy called exhaustive search searches through all possible operators, for all possible nodes. It is computationally expensive, and thus can be applied only to small search spaces.

The beam search selects *n* best operations at each node, and searches only through descriptions that resulted from applying the selected operations. The hill climber algorithm, which is the least computationally expensive, selects a single operation that gives the greatest improvement in terms of the quality of new description. Both beam and hill climber strategies require heuristics to select nodes, while the exhaustive search usually works in an uninformed mode. The exhaustive and beam searches usually use the tentative strategy, while the hill climber uses an irrevocable strategy. The three types of search strategies are visualized in Figure 4. Other search methods, e.g., genetic algorithms, can also be used to perform the search (Goldberg, 1989, Holland et al., 1986).

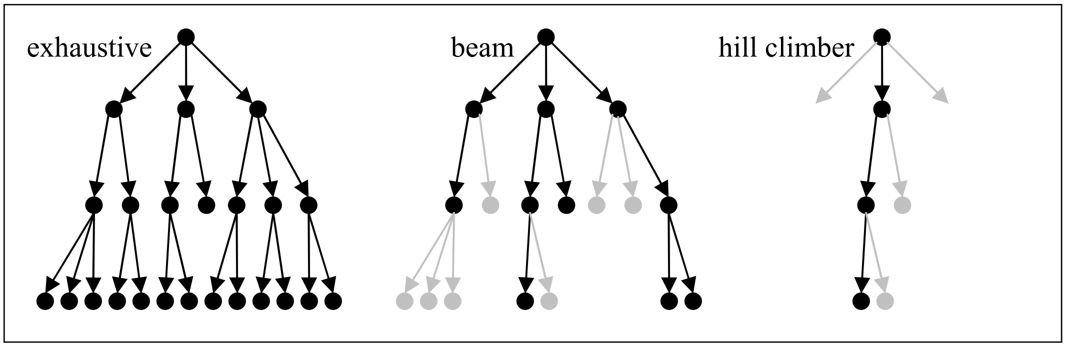


Figure 4. Search strategies

1.1.4 Classification task

A typical application of the rules generated by IL algorithms is classification. A common feature of IL algorithms is their ability to almost perfectly classify the training set, which corresponds to high correctness of the generated rules. However, the true value of the rules generated by an algorithm should be evaluated only by testing them on new, unseen during learning, data. Figure 5 shows how an IL algorithm is used to generate and test a data model (rules) using input data, and how the model is used to perform a classification task. First, input data is divided into disjoint training and testing sets. The training set is used to generate rules, while testing data is used to evaluate validity of the generated rules. Once the rules achieve satisfactory quality level, usually in terms of accuracy of describing data from the test set, they are used to perform classification on data that was not used during the training and testing process.

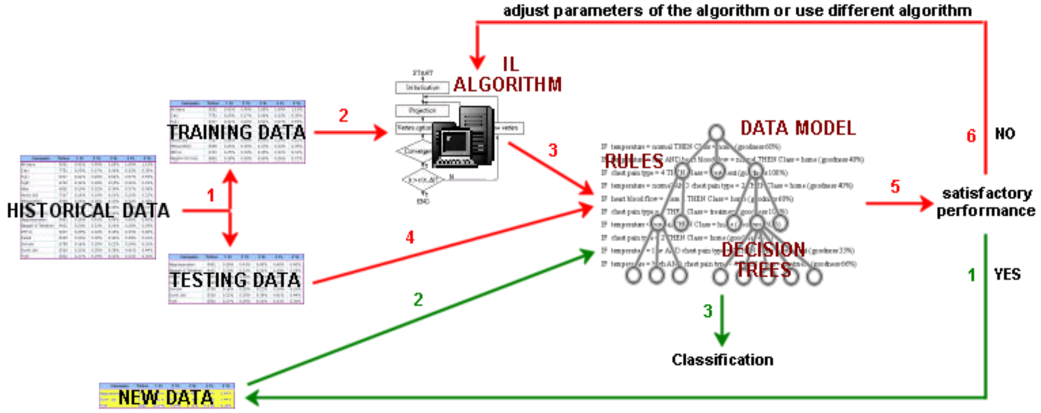


Figure 5. Classification task

1.1.5 Knowledge Representation

There are several models for representation of knowledge, propositional-like, first order logic, and structured representation models (Holsheimer and Siebes, 1994). In this work, we are only concerned with generation of knowledge that utilizes the propositional-like model. The first order logic and the structured representation are characterized by better expressive power, when compared with the propositional-like model. On the other hand, achieving this feature requires substantial increase in complexity of the representation model. The propositional-like representation is characterized by simplicity and offers enough expressive power for use in IL. Since one of the goals of this work is to develop a system that generates simple and easily understandable results, the propositional-like representation was selected.

The propositional-like model uses logic formulas, consisting of attribute-value conditions, which are used in conjunctions and/or disjunctions to express rules. Two most popular propositional-like models are decision trees and production rules.

1.1.5.1 Decision Trees

A decision tree is capable of expressing knowledge about data described by a finite number of classes. The tree consists of nodes and labeled edges. Nodes represent attributes, while edges represent possible values of the attributes. The terminal nodes in the tree, called leaves, represent classes. The tree is used to perform a classification of examples by following a path down the tree, starting from the top (root) node, and descending down by following edges, corresponding to the values of the attributes, until a leaf node is reached. The class value assigned to the leaf node defines classification outcome. The decision tree representation is utilized by decision tree algorithms. An example tree is shown in Figure 6.

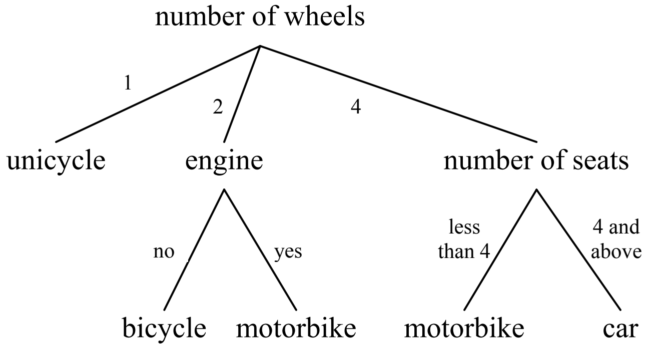


Figure 6. Example decision tree represenation

1.1.5.2 Production Rules

A production rule is capable of expressing knowledge about data described by a finite number of classes. It consists of a structure described by definition 3. The description incorporates a conjunction of conditions on attributes. The main advantages of production rules are their simplicity and modularity, i.e., a single rule can be understood without reference to other rules.

Any decision tree can be converted into a set of production rules (Quinlan, 1987a). A single rule is generated for each leaf node, where class of the rule is defined by a value assigned to the leaf. The rule is generated by following a path that starts at the root node, and ends at the leaf node. The description of a rule consists of selectors that are generated from nodes on the path. An example of a rule generated from tree shown in Figure 6 is:

IF number of wheels = 2 AND engine = no THEN bicycle.

Other types of propositional-like representations include decision lists (Rivest, 1987) and ripple-down rule sets (Holsheimer and Siebes, 1994). These types are more complex than decision tree and production rules, and thus are not considered in this dissertation

1.1.6 Machine Learning and Data Mining

Since modern databases store very large quantities of data, the inference process can be performed only by using computer driven algorithms. Automation of the IL process has been extensively studied in the field of Machine Learning (ML) field (Kodratoff, 1988; Langley, 1996; Mitchell, 1997; Cios et al., 1998). ML is one of the most successful tools of Knowledge Discovery (KD). If we define the KD as a nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns from large collections of data (Fayyad et al. 1996a), then ML is concerned with a Data Mining (DM) step, which is one of the most important steps of the knowledge discovery process (Brachman, R., and Anand, 1996; Fayyad et al., 1996a; Cabena et al., 1998; Cios, 2001; Cios and Kurgan, 2002b). ML can be defined as a field that "is concerned with the question of how to construct computer programs that automatically improve with experience" (Mitchell, 1997), or as "ability of a program to generate a new data structure that is different from an old one, like production if…then… rules from input numerical data" (Cios et al., 1998). Most ML algorithms use induction process to generate the rules (Breiman et al., 1984; Michalski et al., 1986; Clark and Niblett, 1989; Holte, 1993; Quinlan, 1993; Cios and Liu, 1995a; Cios and Liu, 1995b; Cios et al., 1997; Kaufman and Michalski, 1999; Cios and Kurgan, 2001; Cios and Kurgan, 2002a; Kurgan and Cios, 2002a).

The outcome of an inductive ML algorithm is explicit and usually takes form of production rules, or decision trees that are usually converted into rules. One of the reasons why rules generated by ML algorithms are useful is that they are widely used for representing knowledge (e.g., in knowledge-based systems), and can be easily interpreted, learned from, or modified by human experts because of their modularity (Holsheimer and Siebes, 1994). This is one of the main reasons why ML is the preferred DM method in situations where the user needs to understand and validate the generated model, as in the case of medical systems.

1.1.6.1 Why Data Mining and Knowledge Discovery?

DM was brought into attention in 1989 during the IJCAI Workshop on Knowledge Discovery in Databases (KDD) (Piatetsky-Shapiro, 1991). The workshops were continued annually until 1995, when the International Conference on Knowledge Discovery and Data Mining (DMKD) became the most important annual event for DMKD. The framework for DMKD research was outlined in two books: “Knowledge Discovery in Databases“ (Piatesky-Shapiro and Frawley, 1991) and „Advances in Knowledge Discovery and Data Mining“ (Fayyad et al., 1996a). Since then numerous new DMKD conferences such as ACM SIGKDD, SPIE, PKDD, SIAM, regional KDD conferences, DMKD-related workshops, and journals like Data Mining and Knowledge Discovery (1997), Journal of Knowledge and Information Systems (1999), and IEEE Transactions on Knowledge and Data Engineering (1989) have become an integral part of the DMKD field. DMKD applies many artificial intelligence and statistical techniques to the structured data such as databases (Matheus et al., 1993; Fayyad et al., 1996b; Manilla, 1997). The DMKD is an exponentially growing field with strong emphasis on applications (Cios and Kurgan 2002b). The main DMKD research topics include discovery of strong patterns, determination of concept dependencies, selection of representative data examples and most relevant attributes, clustering, and, finally, methods for coping with data deficiencies such as inconsistencies, missing values, and noise. In practice, the terms DM, KD, and DMKD are almost synonymous and are used interchangeably to describe DMKD. For consistency, from now on, we will be using the term DM as the field where this work is situated. This thesis addresses development of a DM system, and thus terms DMKD and KD are less appropriate, since they refer to a discovery process, rather than a methodology for generation of knowledge.

1.2 Motivation

A very important problem of DM is the increasing amount of data that need to be analyzed. Such data are generated on a daily basis by banks, insurance companies, retail stores, medical institutions, research agencies, and on the Internet. This explosion came into being through the ever increasing use of computers, scanners, digital cameras, bar codes, etc. We are in a situation when rich sources of data, stored in databases, warehouses, and other data repositories are readily available. This, in turn, causes big interest of research and industrial communities in DM. We require tools to cope with analysis and understanding of these huge amounts of data. In response, the DM community has developed several successful algorithms over the last few years. A survey that presents a comparison of 43 existing DM algorithms is presented in (Goebel and Gruenwald, 1999).

Still, one of the major difficulties is that many DM algorithms do not scale well to huge volumes of data. A scalable DM algorithm is characterized by linear increase of its runtime with the linear increase of the number of examples in the data, and within a fixed amount of memory. Most of the DM algorithms are not scalable, but there are several examples of tools that do scale well. They include clustering algorithms (Zhang et al., 1996; Bradley et al., 1998; Ganti et al., 1999a), ML algorithms (Shafer et al., 1996; Gehrke et al., 1998), and association rule algorithms (Agrawal and Srikant, 1994; Agrawal et al., 1995; Toivonen, 1996). An overview of scalable DM tools is given in (Ganti et al., 1999b). The most recent approach for dealing with the scalability problem is the Meta Mining (MM) concept. MM generates meta-knowledge from the meta-data generated by DM algorithms (Spiliopoulou and Roddick, 2000). This is performed in two steps. First step is concerned with generation of rules (data models) by a DM algorithm from prepared subsets of the training set. Next, meta-knowledge is generated from the generated rules. In this approach small data models are processed as input data instead of huge amounts of the original data, which greatly reduces the computational overhead (Kurgan and Cios, 2002a).

Another problem addressed in this work is connected with a common practice in the DM field, namely, that the researchers often omit discussion of the complexity of generated models when presenting their results. This leads to the development of algorithms that generate very complex models, which causes difficulty with their understanding. This problem is very important in any domain where the user is required to comprehend and understand the generated model before applying it. A good example is medicine where medical professionals must verify data models before using them in a clinical setting. One of the most straightforward ways to improve understandability of the generated models is to generate them using easy to understand representations, e.g., rules or trees, and to generate models that are as compact as possible.

Finally, the impact of this work can be summarized by putting it into a broader context. IDC (http://www.idcresearch.com/), a well known provider of technology intelligence and industry analysis, estimates that the DM tools market will reach $1.85 billion in 2006. In 1998, Evangelos Simoudis of IBM, predicted that "within five years, [data mining] will be as important to running a business as the business systems are today".

1.3 Goal of the Research

The goal of this research was to design and implement a novel DM system that applies supervised inductive ML for analysis of supervised data. The system generates data models, in terms of rules, from supervised data. Referring to the discussion presented above, the designed system is characterized by (Kurgan and Cios, 2002a; Kurgan and Cios, 2003a):

- ability to generate very compact models. The system generates a set of rules that describe target concepts from supervised data. It generates not only small number of rules, but the generated rules are very compact, in terms of the number of selectors in their description. This is an important feature for a decision maker who needs to evaluate and understand the rules. Also, by the Ockham's razor principle, generation of short descriptions results not only in improved understandability, but also in probably better validity of the rules.
- liner complexity, which is later shown both theoretically and experimentally. This feature enables the use of the system for problems with megabytes and even gigabytes of data. This is a very important advantage, since most of the ML algorithms, except decision tree algorithms, do not scale well. The MetaSqueezer system belongs to the family of rule algorithms, and as such generates different, complementary to decision tree algorithms, types of rules.

The system is characterized by all features of a modern IL algorithm, i.e., accurate classification, efficient generation of simple rules, and flexibility. It uses a novel architecture based on the MM concept, which is the main reason for achieving high compactness of rules. To the best of our knowledge, the

MetaSqueezer system is the first ML-based system that works employing the MM concept. The system provides advancement in the areas of Inductive Learning, Data Mining, Machine Learning, and Meta Mining. The main fields contributing to the research performed in this dissertation, and their interrelationship are shown in Figure 7.

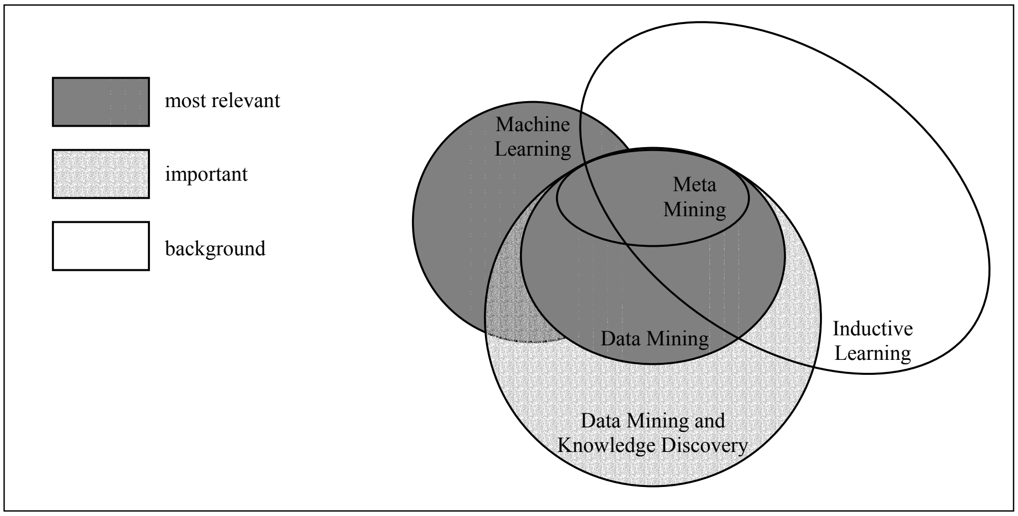


Figure 7. Fields investigated in this dissertation

Chapter 2

2 Related Work

This chapter describes state of the art of Machine Learning and Data Mining fields. First, we provide background information including information about several ML algorithms relevant to the MetaSqueezer system. Next, we introduce necessary information for comparison with existing ML algorithms.

2.1 Introduction

The MetaSqueezer system uses inductive ML techniques to perform generation of rules from input supervised data. Each supervised ML algorithm should have these qualities:

- accurate classification – the generated rules need to accurately classify examples which were unseen during learning,
- simple rules – the generated rules should be compact, since less complex rules are easier to comprehend and are characterized, by the Ockham's razor, by improved validity,
- efficient rule generation – the algorithm must scale up to generate rules for large data sets,
- flexibility – the algorithm should work on wide range of problems. In case of supervised inductive learning, problems are characterized by

type of attributes (discrete numerical, discrete nominal, continuous), and presence of noise and missing values.

The first two qualities are typical for any IL task. The third, however, requires ML algorithms to be scalable. Many early AI algorithms were designed to work only for problems with small number of examples, but were not scalable (Schank, 1991). Presently, the fundamental requirement for any DM and/or ML algorithm is that it must be able to process large amount of data. The first three qualities are typical in design of ML algorithms (Clark and Niblett, 1989), and only state of the art ML algorithms exhibit all four qualities (Esposito et al., 2001, Cios and Kurgan, 2002a).

There are also several other issues specific to ML. Many data sets, used by ML algorithms, contain noise and missing values. Noise can be present in both the predicting and predicted attributes. In the latter case, we have so called false examples. Also, many data sets contain missing information, especially for domains where the data was collected manually. Only some of the ML algorithms are noise and missing data tolerant. A missing data tolerant algorithm can generate the rules from the data that contain missing information. A noise tolerant algorithm can generate rules that do not cover noisy examples. To show the extend of the problem, Redman (Redman, 1998) points out that the data collected by enterprise companies consist of about 1-5% errors, and are often redundant and inconsistent.

The last issue is the format of input data. Some of the data may incorporate different types of attributes. The possible types include nominal, discrete numerical and continuous data. While some of the ML algorithms are able to work with all data types, other algorithms may not be able to work, for example with continuous attributes, which in turn narrows down their application range. Thus, a flexible inductive ML algorithm must be able to overcome all of the above issues. The MetaSqueezer system not only exhibits the first three qualities, but also satisfies the requirement of flexibility.

2.2 Comparing ML Algorithms

Since quality of inductive ML algorithm is characterized by multiple factors, a rigorous method for their assessment is needed.

The accuracy quality requires that an algorithm must be correct and valid. Correctness is exhibited by almost all ML algorithms. It is usually assured by a proper design of the algorithm. Such design must assure that generated rules describe only positive examples, while excluding negative examples. The assessment of validity of rules consists of evaluation of their accuracy while performing classification task on a test set. In this work, a more precise verification test is used to assess validity of generated rules.

The simplicity is assessed by an evaluation of the complexity of generated rules, and is usually measured as length of generated rules. Such simple evaluation is possible, since most of inductive ML algorithms use relatively simple, relational-like knowledge representation models.

The scalability is reflected in the computational complexity of the algorithms. This is often shown using benchmarking tests rather than formal complexity analysis. Although most of inductive ML algorithms do not report theoretical complexity, this work provides both theoretical and experimental scalability evaluation for the system. Since scalability analysis is an experimental-intensive activity, it requires a very precise definition concerning setup and procedure of the performing test. These are explained in the sections below. Finally, flexibility of an inductive ML algorithm is evaluated based on factors that include input attribute types, noise resistance, and missing value resistance.

2.2.1 Assessment of Validity

The assessment of validity of rules is performed by computing accuracy or verification tests while performing classification task with the generated rules on a test set. Two procedures are utilized to perform such test:

- single-split testing. The test is performed by dividing the original data into training and testing sets. The sets are disjoint, and the first one is used to derive data model, and the second to test it. The test results using the accuracy test report only accuracy, while test results using the verification test report accuracy, sensitivity, and specificity.
- k-fold cross validation. In k-fold cross-validation, the data is divided into k subsets of approximately equal size (Efron, 1979). Rules are generated

k times by an IL algorithm, each time leaving out one of the subsets from training, but using only the omitted subset to calculate tests for the algorithm. Similarly, test results using the accuracy test report only accuracy, while test results using the verification test report accuracy, sensitivity, and specificity. The test results report mean values standard deviations of the above criterions, averaging through all k tests. The 10-fold cross validation is the most often used cross validation procedure.

The k-fold cross validation is a more reliable test procedure than the single-split method. It shows the true performance, in terms of validity, of the tested IL algorithm. On the other hand, using the single-split procedure may lead to fitting the generated rules to the test set. This is possible since only one test is performed, which may lead to falsifying the true performance of the tested algorithm.

When comparing an IL algorithm with other algorithms, the choice of one of the above procedures depends on the test procedures done by researches who developed the methods. If they performed the single-split method, the same method must be used.

2.2.1.1 Repositories of Benchmarking Data Sets

There exist several repositories of standard benchmarking data sets, which are widely used by researchers in the IL, ML, and DM communities to perform testing and comparison between different algorithms. Sample repositories include the University of California Irvine (UCI) Machine Learning Repository (Blake and Merz, 1998), the UCI KDD Archive (Hettich & Bay, 1999), and the StatLib project repository (Vlachos, 2000). The repositories store a number of data sets, which are characterized by different properties, such as different attribute types, different number of classes, etc. This enables using them with many different IL algorithms.

Some of standard benchmarking data sets are already prepared for testing procedures. For most of such data sets, they are prepared for single-split test, and thus for the sake of consistency between different tests performed using the same data set, this testing procedure must be used. The system described here was tested using standard data sets downloaded from the above repositories. It uses the same test procedures as the procedures used by researchers who reported their results. Doing this assures reliable comparison with similar algorithms.

2.2.2 Experimental Assessment of Scalability

The scalability of an IL algorithm is often evaluated by showing the algorithm's running time for benchmarking tests. To enable comparison of such test between different algorithms, first the same data sets in the same test configurations must be used. Second, a procedure for computing the running time of algorithms, which were executed on different hardware platforms, must be defined. The assumption is that each of the researchers may have only limited hardware resources, and rather than forcing them to use the same hardware platform, the results are recomputed to accommodate for using different platforms.

The most commonly used methods for direct comparison of running time between algorithms on different hardware configurations are the SPEC benchmarking tests (SPEC, 2001). SPEC benchmark tests contain several programs that perform floating-point or integer computations, which are intended to measure performance of different hardware configurations. For example, the benchmarking results study performed in (Lim, Loh & Shih, 2000), where 33 IL algorithms were compared using standard benchmarking data sets, was performed on three different hardware configurations. By using the SPECint92 benchmarking test, they converted all execution times into the execution time achieved when using the DEC3000 model 300 system (DEC).

This work reports results in the same manner. All of the reported running time results for the DataSqueezer algorithm and MetaSqueezer system were converted into the running time when using the DEC system. This configuration was used to be consistent with the results reported in the above study. Only the results reported for the CAIM algorithm use the original running time, since the entire test was performed on the same hardware configuration. Thus, we note that when evaluating running time results, one should analyze ratio of the running times between algorithms, rather then the exact numbers.

To show how the SPEC tests are used, an example recalculation of running time between the hardware configuration used by the author, and the DEC system, is presented. The hardware configurations used to perform the tests in this work is as follows: Intel Pentium III 800 MHz (I800). The hardware configurations along with the corresponding SPEC test results are given in Table 2.

Table 2. SPEC test results for the computer hardware configurations used in the benchmarking tests

	Workstation	SPECint92 results		Workstation	SPECint95 results
I150	Intel Pro Alder 150MHz	243.9	I150	Intel Pro Alder 150Mhz	6.08
DEC	DEC 3000 Model 300	66.2	I800	Intel Pentium III 800MHz	38.9

To recalculate the running times the following steps are taken:

- The SPECint95 benchmarking test was used since I800 configuration was not reported in the SPECint92 test, where DEC was reported. The Intel Pro Adler 150MHz (I150) system was used as a bridge between the SPECint92 and SPECint95 benchmarking tests.
- The CPU time ratio for I150 using DEC as the reference was calculated as follows: DEC time = (I150 time) * (SPECint92 for I150) / (SPECint92 for DEC) ≈ (I150 time) * 3.5.
- The ratio to transform the time from the I150 to I800 was calculated as: I150 time = (I800 time) * (SPECint95 for I800) / (SPECint95 for I150) ≈ (I300 time) * 6.4.
- Both calculated above ratios were multiplied to calculate the CPU time between DEC and I800. The final formula for the I800 is: DEC time ≈ 22.4 * (I800 time).

Thus, the time achieved on I800 configuration is multiplied by 22.4 to compute the time, which is equivalent to the time achieved using the DEC configuration.

2.3 Evaluation of Inductive Machine Learning Algorithms

Three general categories of inductive ML algorithms are: decision trees, rule algorithms, and their hybrids (Cios et al., 1998). An inductive decision tree algorithm generates rules based on finding regularities in the data through data manipulations that use entropy measures (Shannon, 1948) and involve generation of a decision tree. Examples of decision tree algorithms are CART (Breiman, 1984), ID3 (Quinlan, 1986), C4.5 (Quinlan, 1993), and T1 (Holte, 1993). In contrast, an inductive rule algorithm does not use entropy, and does not generate a tree when generating rules. Representative rule algorithms are the AQ family of algorithms (Michalski et al., 1986; Kaufmann and Michalski, 1999), DBLearn (Cai et al., 1991; Han et al., 1992), KEDS (Rao and Lu, 1993), and DataSqueezer (Kurgan and Cios, 2002a; Kurgan and Cios, 2003a). The hybrid algorithms work by combining entropy-based and non-entropy based approaches. They are represented by CN2 (Clark and Niblett, 1989; Clark and Boswell, 1991), and CLIP family of algorithms (Cios and Kurgan, 2001; Cios and Kurgan 2002a). Table 3, partially taken from (Holsheimer and Siebes, 1994), shows summary of major properties of representative inductive ML algorithms.

Table 3. Major properties of inductive ML algorithms

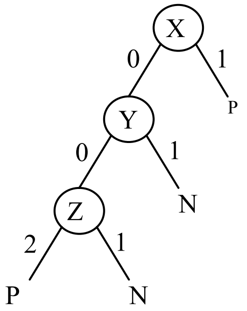
Algorithm	Search Type	Results	Other Features
AQ15	top-down, beam	production rules	only discrete attributes
C4.5	top-down, hill climber	decision tree	all attribute types
CLIP4	top-down, beam	production rules	all attribute types
CN2	top-down, beam	decision lists	all attribute types
DataSqueezer	top-down/bottom-up, hill climber	production rules	all attribute types
DBLearn	bottom-up, hill climber	production rules	only discrete attributes
ID3	top-down, hill climber	decision tree	only discrete attributes
KEDS	top-down, exhaustive	region-equation pair	only discrete attributes
T1	top-down, hill climber	decision tree	1 level tree, all attr. types

2.3.1 Decision Tree Algorithms

The decision tree algorithms are characterized by fast rule generation, high validity of generated results, and high understandability of generated knowledge representation. The T1, ID3, and C4.5, and other ML algorithm that are extensions of them, use the hill climber search technique, which makes them scalable. The CN2 algorithm is characterized by worse scalability, since it applies beam search. Most of decision tree algorithms are also flexible in terms of being able to process both discrete and continuous attributes, and being noise and missing values tolerant. Their limitation is the output knowledge representation, i.e., decision trees. There are two downsides to using decision trees:

- Large trees. Decision tree algorithms tend to grow very large trees for real applications and, thus, may be difficult to interpret by humans (Holsheimer and Siebes, 1994). Only few decision tree algorithms can generate rules that are comparably compact relative to the rules generated by the rule algorithms. They have to apply pruning operation to shorten the rules (Quinlan, 1986; Quinlan 1987b; Mingers, 1989; Esposito et al., 1997). However, pruning adds additional computation overhead and, thus, may worsen scalability.

- Dependent rules. Rules generated by the decision tree algorithms are dependent, i.e., rules share selectors between each other. Since rules are generated by traversing a tree from a leaf to the root node, different rules share the same selectors, as illustrated in Figure 8. This prevents using decision trees in an MM setting, since they would generate meta-rules, which are biased towards the shared selectors.



Rule 1: if X=1 then P
Rule 2: if X=0 and Y=1 then N
Rule 3: if X=0 and Y=0 and Z=1 then N
Rule 4: if X=0 and Y=0 and Z=2 then P

shared selectors:
- rule 2 and 3, and rule 2 and 4: X=0
- rule 3 and 4: X=0, and Y=0

Figure 8. Rule generation using decision trees including sample tree, rules generated from it, and shared selectors

2.3.2 Rule Algorithms

The rule tree algorithms are characterized by slower rule generation, when comparing to decision trees, relatively high validity of generated rules, and high understandability of generated knowledge representation. AQ and CLIP4 apply beam search technique, and thus are not scalable. For example, CLIP4 has $O(s^2)$ complexity, where s is the number of examples in training set (Cios and Kurgan, 2002a). The DBLearn and DataSqueezer apply hill climber search, and their scalability is similar to scalability of the decision tree algorithms. The DataSqueezer algorithm has O(s) complexity (Kurgan and Cios, 2003a), while DBLearn has worse complexity of O(s log s). Also, only DataSqueezer and CLIP4 are flexible in terms of being able to process both discrete and continuous attributes. They are being both noise and missing values tolerant. Other algorithms, i.e., DBLearn and AQ15 can handle only discrete attributes.

The rules generated by rules algorithms do not exhibit dependencies. This enables using them in an MM setting, in contrast to decision tree algorithms. Thus, since the architecture of the MetaSqueezer system is based on an MM concept, a rule algorithm was used to generate rules. The system utilizes DataSqueezer algorithm, since it exhibits best properties among the studied rule algorithms. DataSqueezer is characterized by linear running time, high validity of generated rules, flexibility to handle both discrete and continuous attributes, noisy and missing values resistance, and high understandability of generated rules. These properties are supported by extensive benchmarking comparison shown in the subsequent chapters.

Chapter 3

3 Architecture of the MetaSqueezer System

In this chapter we give a high level overview of the MetaSqueezer system and detailed description of all its components. The Meta Mining concept, which is one of the key concepts used in the MetaSqueezer system, is also described. The description of each component is followed by experimental and theoretical evaluation and comparison with other IL algorithms. A detailed description of the system along with its experimental evaluation is described in the next Chapter 4.

3.1 Introduction

The MetaSqueezer system uses supervised inductive ML methods and MM concept (Kurgan and Cios, 2003a). MM is a generic framework for higher order mining. Its main characteristic is generation of data models called meta-knowledge (often meta-rules) from already generated data models (usually rules, which are called meta-data) (Spiliopoulou and Roddick, 2000). An MM IL system works in two steps. First, it divides the training set subsets and generates a data model for each of them. Next, it takes the generated data models and generates new meta-models from them. Since the MM concept is one of the fundamental technologies utilized to develop the MetaSqueezer system, the following section is used to introduce it to the reader.

3.1.1 Meta Mining

The goal of the MM is to generate rules from rule sets generated by IL algorithms. An IL algorithm, which works based on the MM concept, is shown in Figure 9.

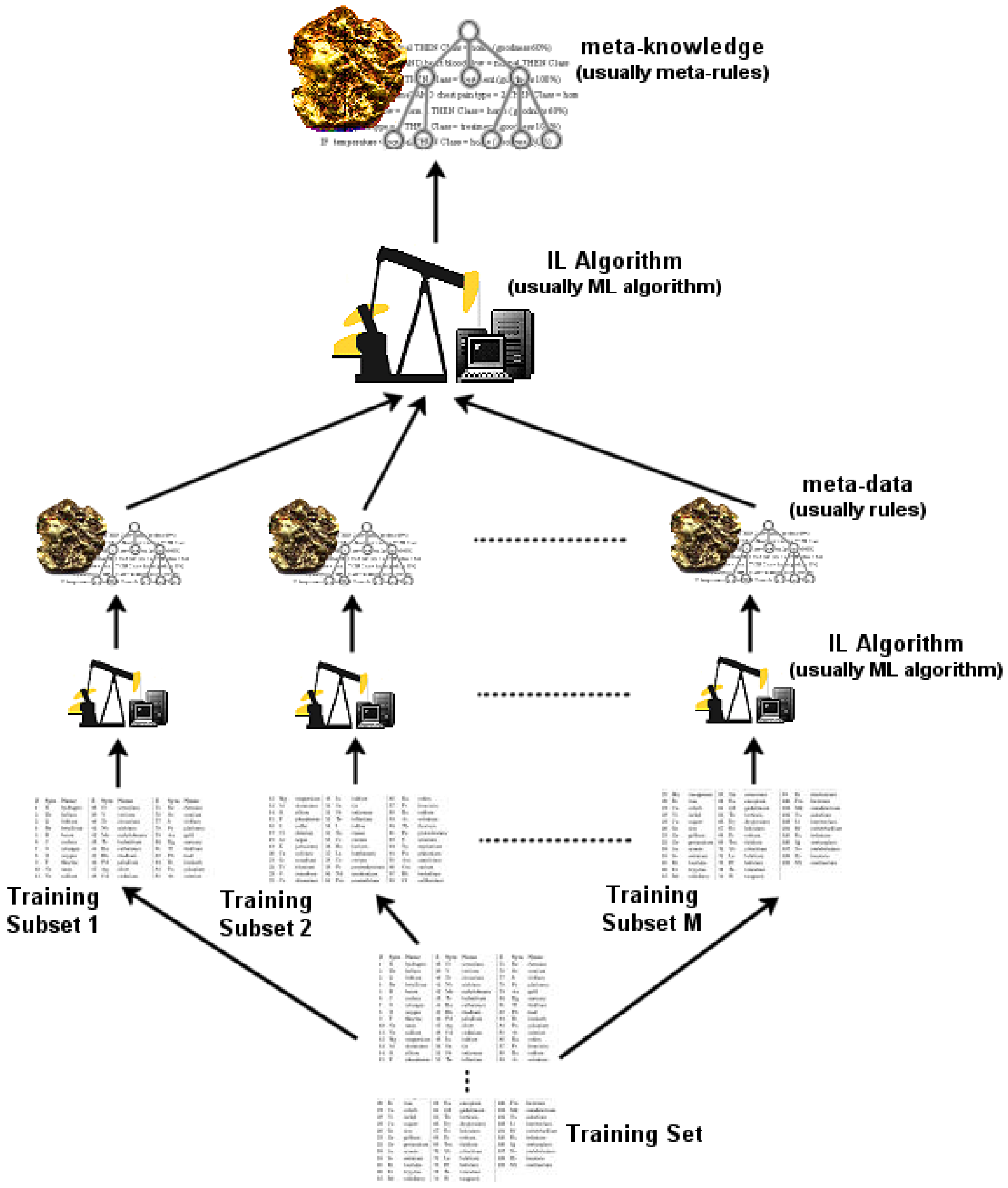


Figure 9. The MM procedure

The MM is based on performing several steps in order to generate a final data model. First, the training set is divided into n subsets. Each of these subsets is fed into an IL algorithm to generate meta-data. The meta-data usually take form of production rules or decision trees. Next, the meta-data is fed into another or the same IL algorithm to generate meta-knowledge. The meta-knowledge is often in a form of meta-rules. The meta-rules may use the same knowledge representation as regular rules, i.e., production rulers, trees, etc., but they express knowledge about the meta-data, rather then the original training set.

There are several advantages to using MM (Spiliopoulou and Roddick, 2000; Kurgan and Cios, 2003a):

- Generation of compact data models. Since an MM based system generates results from data models, they reflect patters that are present in meta-data. Thus, the results of MM are different than results of learning from the original training set. Researchers argue that such results are more suitable for describing knowledge that can be considered interesting by users (Abraham and Roddick, 1999; Spiliopoulou and Roddick, 2000; Roddick and Spiliopoulou, 2002). In many cases, the meta-rules describe the most interesting information since often focus of interest is directed to finding the information that is a confluence of rules which describe a small subset of characteristics about the data (Abraham and Roddick 1999).

- Scalability. The learning algorithms are applied to small sets of data, i.e., subsets of the training set and meta-data, instead of to the huge amounts of data stored in the training set. This results in reduction of computational time for the systems that utilize non-linear algorithms to generate data models.
- User-friendliness. Some argue that meta-mining is characterized by improved tractability of generation of data models and improved ease of finding changes in the data (Spiliopoulou and Roddick, 2000).

Although using MM has the advantage of reducing computational overhead and providing compact data models, so far it received little attention in the DMKD community.

One of the most natural application areas for MM is the temporal KD where rules can be generated for particular time intervals and then meta-knowledge can be discovered using MM. Instead of directly learning from the entire temporal training data set, its parts that describe particular time intervals are learned to generate meta-rules, and later used to generate the meta-knowledge. The MM approach was applied to temporal association rule algorithms (Rainsford and Roddick 1999). The MM in case of the temporal KD performs meta-knowledge learning, which consists of the following steps (Abraham and Roddick, 1997):

1. rule set generation; In this step, individual snapshots of the data are selected and learned from, and generated rules (meta-data) are

collected into separate rule sets (timestamped for identification). The generated rule sets are of the same type, and are generated using the same settings of an IL algorithm to ensure full compatibility.

2. input preparation; In this step, generated rule sets are converted into a consistent format to facilitate rule processing.
3. meta-rule set generation; In this step, a separation algorithm that takes two adjacent, in a temporal sense, rule sets as input, compares them and produces four categories of meta-rules: new, retired, unchanged, and changed rules (some of which may be empty).
4. processing of categories; In this step, each meta-rule category is processed individually to derive general characterizations for the contents of each of the four (or possibly only some selected, e.g. new and expired) categories.

Using MM in this setting results in generation of meta-knowledge that describes changes in the data rather than the data itself. The above temporal MM concept was extended to enable incremental discovery of meta-rules (Abraham and Roddick 1999).

The system described here can be used for generation of data models from any supervised data. In particular, the system can be used for analysis of ordered data. Such data can be divided into subsets using one of the data attributes. One example of such data is temporal data. Since the MetaSqueezer system does not utilize any temporal information, it uses the MM concept to generate regular, non-temporal production rules. Sample applications of the system include analysis of medical data, consumer transactions data, business decision making data, stock market data and many others.

3.2 Overview of the MetaSqueezer System

The MetaSqueezer system uses an inductive ML algorithm to generate meta-rules from supervised data (Kurgan and Cios, 2003a). The system generates the meta-knowledge in a series of these steps (Kurgan and Cios, 2003a):

- Preprocessing:
    - the training set is validated by repairing or removing incorrect records, and marking unknown values,
    - the validated data is transformed into the form suitable for further processing. In case of the MetaSqueezer system, it is a single relational table where a separate column holds an attribute that is used to divide the data. The classes are generated for each data record. They are usually derived from one of the attributes,
    - continuous attributes, of the transformed data, are discretized (Cios et al., 1998) by a supervised discretization algorithm CAIM (Kurgan and Cios, 2001; Kurgan and Cios, 2002b; Kurgan and Cios, 2003b),
    - the data is divided into subsets using the prepared attribute;

- Data Mining
    - meta-data in the form of production rule sets are generated from data for each of the defined subsets. The rules are generated using supervised inductive ML algorithm called DataSqueezer (Kurgan and Cios, 2002a; Kurgan and Cios, 2003a),
    - for every set of rules a rule table is created. It stores the generated rules in the format that is identical to the format of the original input data, for detailed explanation see section 3.3.1. Each table stores meta-data about the input data from one of the data subsets.
- Meta Mining
    - meta-rules are generated from the rule tables. First, all the rule tables are concatenated into a single table. Next, the meta-rules are generated by applying the DataSqueezer algorithm to the combined rule table. The meta-rules describe the most important patterns associated with defined classes over the entire original training set.

The MetaSqueezer system uses the same ML algorithm, called DataSqueezer, to generate both meta-data and meta-knowledge. This is because of specific properties of the DataSqueezer algorithm, which are described in the subsequent sections. The system is characterized by high accuracy and good scalability because of its linear complexity, high compactness of generated meta-knowledge, and flexibility. It works with nominal, and discrete and continuous numerical attributes. It also handles data with missing values and noise. The above properties are discussed and shown in the subsequent sections.

Below, a detailed description of the system is given. Chapter 4 describes components of the system, i.e., DataSqueezer and CAIM algorithms, together with evaluation and comparison with other state of the art ML algorithms. Chapter 5 describes the MetaSqueezer system. It also provides an evaluation and comparison of the system with other state of the art ML systems. Finally, Chapter 6 is devoted to description of application of the system into a real-life problem concerning analysis of cystic fibrosis data.

3.3 The DataSqueezer Algorithm

The DataSqueezer algorithm constitutes core element of the MetaSqueezer system (Kurgan and Cios, 2002a; Kurgan and Cios, 2003a). It is utilized to generate meta-data during the DM step and the meta-rules in the MM step. DataSqueezer is an inductive ML algorithm that generates production rules applying generalization operations, and irrevocable, informed hill climber search. The algorithm applies a bottom-up, followed by top-down approach to generate the rules. Next, we provide nomenclature necessary to give the pseudocode of the algorithm, the pseudocode, and prose explanation of the algorithm.

3.3.1 The Algorithm

Let us denote the input data set by S. The sets of positive examples, S_P, and negative examples, S_N, must satisfy these three properties: $S_P \cup S_N = S$, $S_P \cap S_N =$

$\varnothing$, $S_N \neq \varnothing$, and $S_P \neq \varnothing$. Examples are described by a set of K attribute-value pairs: $e = \wedge_{j=1}^{K}[a_j \# v_j]$, where a_j denotes j^th^ attribute with value $v_j \in d_j$ (domain of values of j^th^ attribute), # is a relation (=, <, ≈, ≥, etc.), and K is the number of attributes (Michalski, 1973; Michalski, 1983b). In case of the DataSqueezer algorithm the relation is equality. An example e consists of set of selectors $s_j = [a_j = v_j]$. The DataSqueezer algorithm generates production rules in the form of: IF (s_1 AND … AND s_m) THEN $class = class_i$, where $s_i = [a_j = v_j]$ is a single selector.

We define S_P and S_N as tables where rows represent examples and columns correspond to attributes. Table of positive examples is denoted as *POS* and the number of positive examples by N_{POS}, while the table and the number of negative examples as *NEG* and N_{NEG}, respectively. The *POS* and *NEG* tables are created by inserting all positive and negative examples, respectively, where examples are represented by rows and attributes by columns. Positive examples from the *POS* table are described by the set of values: $pos_i[j]$ where j=1,…,K, is the column number, and i is the example number (row number in the POS table). The negative examples are described similarly by a set of $neg_i[j]$ values. The DataSqueezer algorithm also uses tables that store intermediate results (G_{POS} for *POS* table, and G_{NEG} for *NEG* table), which are composed of K columns. Each cell of the G_{POS} table is denoted as $gpos_i[j]$, where i is a row number and j is a column number, and similarly for G_{NEG} table is denoted by $gneg_i[j]$. The G_{POS} table stores reduced subset of the data from *POS*, and G_{NEG} table stores reduced subset of the data from *NEG*. The meaning of reduction is explained later. Both, the G_{NEG} and G_{POS} tables have an additional, (K+1)th column that stores number examples from *NEG* and *POS*, which a particular row in G_{NEG} and G_{POS} describes, respectively. Thus, for example $gpos_2[K+1]$ stores number of examples from *POS*, which are described by the 2nd row in G_{POS} table. Figure 10 shows the pseudocode of the DataSqueezer algorithm (Kurgan and Cios, 2003a).

```
Given:   POS, NEG, K (number of attributes), S (number of examples)
Step1.
1.1      Initialize $G_{POS}$ = []; i=1; j=1; k=1; tmp = $pos_1$;
1.2.1    for k = 1 to K                                    // for all attributes
1.2.2      if ($pos_j[k] \neq tmp[k]$ or $pos_j[k]$ = '*')
1.2.3        then tmp[k] = '*';                            // '*' denotes missing value
1.2.4    if (number of non missing values in tmp $\geq$ 2)
1.2.5        then $gpos_i$ = tmp; $gpos_i[K+1]$ ++;
1.2.6      else i ++; tmp =  $pos_j$;
1.3      set j++; and until j $\leq N_{POS}$ go to 1.2.1
1.4      Initialize $G_{NEG}$ = []; i=1; j=1; k=1; tmp = $neg_1$;
1.5.1    for k = 1 to K                                    // for all attributes
1.5.2      if ($neg_j[k] \neq tmp[k]$ or $neg_j[k]$ = '*')
1.5.3        then tmp[k] = '*';                            // '*' denotes missing value
1.5.4    if (number of non missing values in tmp $\geq$ 2)
1.5.5        then $gneg_i$ = tmp; $gneg_i[K+1]$ ++;
1.5.6      else i ++; tmp =  $neg_j$;
1.6      set j++; and until j $\leq N_{NEG}$ go to 1.5.1
Step2.
2.1      Initialize RULES = []; i=1;                       // where $rules_i$ denotes $i^{th}$ rule stored in RULES
2.2      create LIST = list of all columns in $G_{POS}$
2.3      within every column of $G_{POS}$ that is on LIST, for every non missing value a from selected column k
         compute sum, $s_{ak}$, of values of $gpos_i[K+1]$ for every row i, in which a appears (multiply every $s_{ak}$, by the
         number of values the attribute k has)
2.4      select maximal $s_{ak}$, remove k from LIST, add "k = a" selector to $rules_i$
2.5.1    if $rules_i$ does not describe any rows in $G_{NEG}$
2.5.2        then remove all rows described by $rules_i$ from $G_{POS}$, i=i+1;
2.5.3          if $G_{POS}$ is not empty go to 2.2, else terminate
2.5.4      else go to 2.3
Output:  RULES describing POS
```

Figure 10. The pseudo-code of the DataSqueezer algorithm

The first step of the algorithm works in a bottom-up manner. It starts with the most specific hypotheses, which cover individual examples. Next, it applies generalization operator to generalize them until they incorporate two or more selectors. This is performed for both positive and negative data. In the second step, the algorithm applies a search procedure that generates rules from the generalized hypotheses. Each rule is generated in a top-down manner, since the search starts with a single selector, and uses specialization by adding additional selectors until the rule covers only positive examples, from the generalized positive hypotheses, and none of the negative examples, from the generalized negative hypotheses.

The rule generation mechanism used by the DataSqueezer algorithm is based on the inductive learning hypothesis (Mitchell, 1997). It states that any hypothesis found to approximate the target function (defined by class attribute) well, over a sufficiently large set of training examples, will also approximate the target function well over other unobserved examples. Based on this assumption, step 1 of the algorithm performs data reduction via use of the prototypical concept learning. Step 1 is very similar to the Find S algorithm by Mitchell (Mitchell, 1997). For comparison, the Find S algorithm is shown in Figure 11. DataSqueezer performs data reduction to generalize information stored in the original data. The data reduction is performed by application of generalization operator in linear, with respect the training set, manner. First, a very specific hypothesis that covers an example is generated, and next it is generalized by using next and the following examples. The data reduction is performed separately for both positive and negative data. Step 2 of the algorithm generates rules by performing irrevocable, informed hill climber search starting with an empty rule, and adding selectors until the termination criterion fires. The max depth of the search is equal to the number of attributes. Next, the examples covered by the generated rule are removed from the G_{POS} table and the process is repeated until all examples are covered by generated rules.

```
Find S Algorithm
1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x
   - For each attribute constraint $a_i$ in h
     IF the constraint $a_i$ in h is satisfied by x THEN do nothing
     ELSE replace $a_i$ in h by the next more general constraint that is satisfied by x
3. Output hypothesis h
```

Figure 11. The Find S algorithm

The Find S algorithm has several problems with the convergence to a correct hypothesis (Mitchell, 1997). The DataSqueezer algorithm uses step 2 to resolve them. Step 2 assures convergence to the correct and consistent with training set hypothesis by using negative data. Each rule, while being generated, is checked versus the set of generalized negative hypotheses. If the rule covers any data in this set, a new selector is added to the rules making it more specific, and thus able to better distinguish between positive and negative data. Only rules that can describe positive data and none of the negative data are accepted. Thus, the issue of inconsistency of generated hypotheses is resolved. The main properties of the DataSqueezer algorithm are summarized in Table 4.

Table 4. Major properties of the DataSqueezer algorithm

Search Process	Search Type	Results	Other Features
top-down in step1 bottom-up in step 2	irrevocable hill climber	production rules	all attribute types, noise and missing values resistant

For multi-class problems, the DataSqueezer algorithm generates a separate set of rules for every class, each time generating rules that describe the currently chosen (positive) class. The DataSqueezer algorithm utilizes CAIM algorithm to perform front-end discretization of continuous attributes (Kurgan and Cios, 2001; Kurgan and Cios, 2002b; Kurgan and Cios, 2003b).

The following section provides an answer to the question why the DataSqueezer was chosen, among other ML algorithms, to be incorporated into the MetaSqueezer system. The DataSqueezer algorithm was chosen because of its five following characteristic features (Kurgan and Cios, 2002a; Kurgan and Cios, 2003a):

- **it generates production rules that involve no more than one selector per attribute**. The number of selectors of a rule generated by the DataSqueezer algorithm is bounded by the number of attributes in the data. Similar property is exhibited only by decision tree algorithms. The rule and hybrid algorithms generate rules that may involve multiple selectors per single attribute.

  This property enables storing generated rules in a table that has identical structure with the original data table. The following example of patients who are either directed to go home or to undergo a treatment, based on some test results, is used to present this property. Every patient is described by three attributes: temperature, heart blood flow results, and chest pain type. The attributes can take on the following values: temperature = {normal, low, high}, heart blood flow = {normal, low, high}, and chest pain type = {1, 2, 3, 4}. Based on the values of the attributes, a decision is made for each patient. We note that the example is used only to show how the algorithm is working, while the algorithm is able to handle much more complex and larger training sets. The DataSqueezer algorithm is used to generate a set of rules that describe how the decision was made based on some historical data. A sample historical data is shown in Table 5.

Table 5. Sample training set for the DataSqueezer algorithm

decision	temperature	heart blood flow	chest pain type
home	low	normal	2
home	low	normal	3
home	normal	normal	3
home	normal	low	2
home	normal	low	1
treatment	low	high	4
treatment	low	low	4
treatment	high	normal	4

DataSqueezer generates the following three rules for the above data: “IF temperature = normal THEN home”, “IF temperature = low AND heart blood flow = normal THEN home”, and “IF chest pain type = 4 THEN treatment”. The rules describe historical data and at the same time show decision patterns. The three rules can be stored in a rule table shown in Table 6. This table has identical format with the table shown in Table 5.

Table 6. Sample rule table generated by the DataSqueezer algorithm

decision	temperature	heart blood flow	chest pain type
home	normal	*	*
home	low	normal	*
treatment	*	*	4

The “*” symbol stands for missing value, and is used to denote attributes that are not used in the rules. The rule table can be used as input to the same algorithm that was used to generate the rules. This enables using the DataSqueezer algorithm in an MM setting by running it again with the generated rules (meta-data) as its input to produce the meta-rules. The meta-rules describe patterns exhibited in the meta-data, and thus they provide a well-focused and compact knowledge about the classes.

- **it generates rules that are very compact in terms of the number of used selectors**. Experimental results, shown later, indicate that DataSqueezer generates rules which involve small number of selectors. This assures that the input data for the MM step of the MetaSqueezer system are very compact and focused on describing the classes. This, in turn, improves the results generated by the system. For comparison, decision trees tend to grow very large and, thus, are difficult to interpret by humans, unless pruning is used to shorten the rules. For details see section 2.3.1.
- **it can handle data with large number of missing values**. The rule generation procedure used by the DataSqueezer algorithm assures that it can cope with data that has large number of missing values. This property is especially valuable in an MM setting. The meta-data generated by the DataSqueezer algorithm contains many missing values since it provides a compact description of the original data. It omits

some attributes and selectors, used in the original data, to provide compact meta-data. The omitted values are substituted by missing values in the rule tables, which are used as an input to the MM step.

The DataSqueezer algorithm uses only complete portion of the data. In other words it uses all available information while ignoring missing values. It uses all examples from the original data, even the examples that contain missing values, by analyzing their complete values.

- **it generates independent rules**. Rules generated by the DataSqueezer are independent of each other, i.e., the rule generation mechanism does not allow for sharing of selectors between the rules. To compare, the decision trees generate dependent rules, see section 2.3.1. If decision trees were to be used in an MM setting, the generated meta-rules would be biased towards the shared selectors. Thus, using decision trees can results in omitting some of important patterns in the generated meta-rules.
- **it has linear complexity**. Complexity analysis of the algorithm shows that it is liner in respect to the number of examples in the data sets, i.e., O(m), where m is the number of examples. The theoretical complexity analysis of the algorithm is shown in section 3.3.1.1.

The above discussion rules out decision tree algorithms as potential candidates for incorporation in the MetaSqueezer system since they tend to grow large trees as well as dependent rules. Since the DataSqueezer algorithm is characterized by the best features among the state of the art rule and hybrid algorithms, see section 2.3.2, it was chosen to be used in the system.

3.3.1.1 Theoretical Complexity Analysis

In what follows, complexity of the DataSqueezer algorithm is estimated. First we start by defining assumptions:

- s is number of examples, k is number of attributes, r is the number of generated rules, and c is the number of classes in the problem,
- length of the *RULES* vector is k,
- size of all *POS* and *NEG* matrices is $kO(s)$,
- $s>>k$, since this is required to apply ML algorithm,
- the r, c, and k are small constants. These constants were used in the analysis to provide general complexity estimation, but the final complexity is a function of s.

To estimate complexity of the entire algorithm, we break the process into determination of the complexity for particular steps of the algorithm:

1. complexity of the initialization (line “Given” from the code from Figure 10)

   $k\mathrm{O}(s)$ to derive POS matrix

   $k\mathrm{O}(s)$ to derive NEG matrix

   Thus, the total complexity of the initialization is: $k\mathrm{O}(s)$.

2. complexity of STEP 1 (lines 1.1-1.6 from the code from Figure 10)

   Line 1.1: O(1)

   Line 1.2.1: O(k) and applies to lines 1.2.2, and 1.2.3

Line 1.2.2:	$O(1)$	
Line 1.2.3:	$O(1)$	
Lines 1.2.4-1.2.6:	$O(1)$	
Line 1.3:	$O(s)$	and applies to lines 1.2.1-1.2.6
Line 1.4:	$O(1)$	
Line 1.5.1:	$O(k)$	and applies to lines 1.5.2, and 1.5.3
Line 1.5.2:	$O(1)$	
Line 1.5.3:	$O(1)$	
Lines 1.5.4÷1.5.6:	$O(1)$	
Line 1.6:	$O(s)$	and applies to lines 1.5.1-1.5.6

Thus total estimated complexity of the STEP 1 is: $O(1) + O(s) \bullet [\ O(k) \bullet [\ O(1) + O(1)\ ] + O(1) + O(1) + O(1)\ ] + O(1) + O(s) \bullet [\ O(k) \bullet [\ O(1) + O(1)\ ] + O(1) + O(1) + O(1)\ ] = O(1) + O(ks) + O(s) + O(s) + O(s) + O(1) + O(ks) + O(s) + O(s) + O(s) = O(ks)$.

3. complexity of STEP 2 (lines 2.1-2.5.4 from the code from Figure 10)

Line 2.1:	$O(1)$	
Line 2.2:	$O(1)$	
Line 2.3:	$O(ks)$	one sweep through G_{POS} is sufficient
Line 2.4:	$O(k)$	selection of max s_{ak} is precomputed in 2.3
Line 2.5.1:	$O(ks)$	one sweep through G_{NEG} is required
Line 2.5.2:	$O(s)$	

Line 2.5.3: O(r) and applies to lines 2.2-2.5.4, since this line will execute “go to 2.2” r times

Line 2.5.4: O(k) and applies to lines 2.3-2.5.4, since the longest rules has k selectors

Thus the total complexity of the STEP 2 is: O(1) + O(r) • [O(1) + O(k) • [O(ks) + O(k) + O(ks) + O(s)]] = O(1) + O(r) • [O(1) + O(k^2s) + O(k^2) + O(k^2s) + O(ks)] = O(1) + O(r) + O(rk^2s) + O(rk^2) + O(rk^2s) + O(rks) = O(rk^2s).

The complexity of the entire algorithm is estimated as a sum of complexities for each of the algorithm’s steps as: kO(s) + O(ks) + O(rk^2s) = O(rk^2s).

The above estimation concerns generation of rules for one class. The complexity of generation of rules for the problems with c classes is cO(rk^2s). Since the number of generated rules r and number of classes c are usually small constants, we can estimate that the expected running time of the algorithm is O(k^2s). Additionally, since number of attributes k is also usually a small constant, the expected running time of the algorithm is estimated as O(s). This argument shows that the DataSqueezer algorithm is linear. Linear complexity of the DataSqueezer algorithms proves that it is scalable and can therefore be used with data sets of large size.

3.3.2 Experimental Evaluation

The DataSqueezer algorithm was extensively benchmarked to evaluate its validity, simplicity, efficiency and flexibility. It was tested on 20 data sets. The data sets were obtained from the University of California Irvine (UCI) Machine Learning Repository (Blake and Merz, 1998), the UCI KDD Archive (Hettich & Bay, 1999), and from the StatLib project data sets repository (Vlachos, 2000). The reason for using standard benchmarking data sets is that it enables direct comparison of performance of the DataSqueezer algorithm with other ML algorithms that generate similar results. A detailed description of the data sets is given in Table 7 Both, percent of missing values and percent of inconsistent examples refer to already discretized training sets.

The benchmarking tests are characterized by diversity of training sets, including their size, both in terms of number of examples and attributes, number of classes, attribute types, and amount of missing values and noise in the data. The range of the tests is characterized by:

- the size of training data sets: between 151 and 200K examples,
- the size of testing data sets: between 15 and 565K examples,
- the number of attributes: between 5 and 61,
- the number of classes: between 2 and 10,
- the attribute types: nominal, discrete numerical, and continuous numerical,
- the percentage of examples with missing values: between 0 and 52.3,

- the percentage of inconsistent, noisy examples: between 0 and 66.3.

Table 7. Description of data sets used for benchmarking of DataSqueezer algorithm

set	size	# classes	# attrib.	test data	% ex. with missing	% inconsistent examples	# subsets
bcw	699	2	9	10CV	2.3	0	4
bld	345	2	6	10CV	no	62.2	3
bos	506	3	13	10CV	no	22.9	3
cmc	1473	3	9	10CV	no	53.6	7
dna	3190	3	61	1190	no	0	8
hea	270	2	13	10CV	no	1.9	3
led	6000	10	7	4000	no	66.3	10
pid	768	2	8	10CV	no	60.9	4
sat	6435	6	37	2000	no	3.6	10
seg	2310	7	19	10CV	no	0.8	6
smo	2855	3	13	1000	no	36.5	4
thy	7200	3	21	3428	no	1.8	6
veh	846	4	18	10CV	no	6.7	4
vot	435	2	16	10CV	no	0	3
wav	3600	3	21	3000	no	0	3
tae	151	3	5	10CV	no	56.1	2
adult	48842	2	14	16281	7.4	31.9	10
cid	299285	2	40	99762	52.3	0.01	10
forc	581012	7	54	565892	0	6.4	5
mush	8124	2	22	2441	29.7	0	5

This diversity of the tests ensures that evaluation of the algorithm, which is performed based on comparison with results achieved by other state of the art inductive ML algorithms, is comprehensive and strong.

3.3.3 Comparison of DataSqueezer with Other Algorithms

The tests compare accuracy of the rules, number of rules and selectors, and execution time. The DataSqueezer algorithm was compared to CLIP4 algorithm (Cios and Kurgan, 2002a), and 33 other inductive ML algorithms, for which the results were published in (Lim et al., 2000). This study reports results on the first 16 data sets described in Table 7. The remaining 4 data sets were chosen because of their larger size, and incorporation of larger amount of missing values. The last, *cid*, data set was also used to perform experimental complexity analysis.

3.3.3.1 Accuracy

The tests, see Table 8, show verification test results for the DataSqueezer and other ML algorithms (Kurgan and Cios, 2003a). For the 33 algorithms maximum and minimum accuracy, after (Lim et al., 2000), is reported.

The mean accuracy of the DataSqueezer algorithm for the first 16 data sets is 75.4%. To compare, the POLYCLASS algorithm achieved the highest mean accuracy of 80.5% (Kooperberg et al. 1997). (Lim et al., 2000) calculated statistical significance of error rates. It shows that the difference between the mean accuracies of two algorithms is statistically significant at the 10% level if they differ by more than 5.9 %. Close analysis of the results indicates that the DataSqueezer algorithm's accuracy is within the range of the best ML algorithms.

Table 8. Accuracy results for the DataSqueezer, CLIP4, and the other 33 ML algorithms

set	Reported accuracy (Lim et al., 2000)		CLIP4 accuracy (Cios and Kurgan, 2002a)	DataSqueezer		
	max	min		mean accuracy	mean sensitivity	mean specificity
bcw	97	91	95	94	92	98
bld	72	57	63	68	86	44
bos	78	69	71	70	70	88
cmc	57	40	47	44	40	73
dna	95	62	91	92	92	97
hea	86	66	72	79	89	66
led	73	18	71	68	68	97
pid	78	69	71	76	83	61
sat	90	60	80	80	78	96
seg	98	48	86	84	83	98
smo	70	55	68	68	33	67
thy	99	11	99	96	95	99
veh	85	51	56	61	61	88
vot	96	94	94	95	93	96
wav	85	52	75	77	77	89
tae	77	31	60	55	53	79
MEAN	**83.5**	**54.6**	**74.9**	**75.4**	74.6	83.5
set	algorithm (accuracy) (reference)			mean accuracy	mean sensitivity	mean specificity
adult	NBTree (84) (Kohavi, 1996) C4.5 (84.5), C4.5-auto (85.5), Voted ID3-0.6 (84.4), T2 (83.2), 1R (80.5), CN2 (84), HOODG (83.2), FSS Naive Bayes (**86**), IDTM (85.5), Naive-Bayes (83.9), NN-1 (78.6), NN-3 (79.7), OC1 (85) (Blake & Merz, 1998)			82	94	41
cid	C4.5 (95.2), C5.0 (95.3), C5.0 rules (95.3), C5.0 boosted (**95.4**), Naïve-Bayes (76.8) (Hettich & Bay, 1999)			91	94	45
forc	[NN-backprop (**70.0**), Linear Discriminant Analysis (58.0)] (Blackard, 1998)			55	56	90
mush	[C4.5 (**100**), NBTree (96.5)] (Kohavi, 1996) [STAGGER (95)] (Schlimmer, 1987) [HILLARY (95)] (Iba, Wogulis & Langley, 1988)			100	100	100
MEAN	mean best: **87.9** means worst: **77.1**			**82.0**	86.0	69.0

We also note that since the mean sensitivities and specificities achieved by the algorithm have high and comparable values, the generated rules describe correctly all classes in the data. The rules are not biased towards describing, for example, only classes that are described by majority of examples.

The results achieved by the DataSqueezer for the latter four data sets also place it among best ML algorithms. It achieves accuracies comparable to the results obtained by other best ML algorithms for the *adult*, *cid*, and *mush* data sets.

The results show that DataSqueezer generates very accurate rules, which means that it is characterized by high validity.

3.3.3.2 Simplicity and Efficiency

The tests, see Table 9, also show number of rules, number of selectors, and execution time for the DataSqueezer algorithm, and the other ML algorithms (Kurgan and Cios, 2003a). Similarly, as for the accuracy tests, the algorithm is compared with results achieved by 33 algorithms reported in (Lim et al., 2000), and with results achieved by the CLIP4 (Cios and Kurgan, 2002a). For the 33 algorithms, the median number of rules (the authors reported the number of tree leaves for 21 decision tree algorithms) and the maximum and minimum execution time, as it was reported by the authors, is given. Additionally, for the DataSqueezer and CLIP4, the number of selectors per rules is reported. The last measure enables direct comparison of complexity of generated rules.

The mean number of rules generated by the DataSqueezer algorithm is 21.3. In (Lim et al., 2000) the median number of tree leaves, which is equivalent to the number of rules, for the 21 tested decision tree algorithms was reported as 17.8. The number of rules generated by the CLIP4 algorithm is 16.8. The number of rules generated by the DataSqueezer algorithm is comparable to the reported results. For the latter four data sets, the algorithm also achieves low number of generated rules.

Table 9. Number of rules and selectors, and running time results for the DataSqueezer, CLIP4, and the other 33 ML algorithms

set	Reported accuracy (Lim et al., 2000)			CLIP4 (Cios and Kurgan, 2002a)				DataSqueezer			
	mean CPU time		median # of leaves					mean time [s]	mean # rules	mean # select	# select/ rule
	min [s]	max [h]		mean time [s]	mean # rules	mean # selectors	# select /rule				
bcw	4s	2.7	7	5.1	4.2	121.6	29.0	0.2	4.5	12.8	2.8
bld	5s	1.5	10	6.6	9.7	272.4	28.1	0.1	3.4	14.0	4.1
bos	9s	5.5	11	35.8	10.5	133.5	12.7	0.4	19.8	107	5.4
cmc	12s	23.9	15	46	8	60.7	7.6	1.3	20.2	70.5	3.5
dna	2s	475.2	13	662.8	8	90	11.3	12.5	39.0	97.0	2.5
hea	4s	3.3	6	2.2	11.6	192.3	16.6	0.1	4.7	17.1	3.6
led	1s	12.4	24	166.4	41	189	4.6	3.8	51	194	3.8
pid	7s	2.5	7	5.8	4	64.1	16.0	0.2	1.8	8.0	4.4
sat	8s	73.2	63	3696.7	61	3199	52.4	21.3	57	257	4.5
seg	28s	75.6	39	614.6	39.2	1169.9	29.8	6.1	57.3	219	3.8
smo	1s	3.8	2	90.5	18	242	13.4	1.1	6	12	2.0
thy	3s	16.1	12	164.2	4	119	29.8	1.3	7	28	4.0
veh	14s	14.1	38	45.3	21.3	380.7	17.9	1.2	23.7	80.2	3.4
vot	2s	25.2	2	4.4	9.7	51.7	5.3	0.1	1.4	1.6	1.1
wav	4s	4.3	16	43.1	9	85	9.4	0.4	22	65	2.9
tae	6s	10.2	20	0.7	9.3	273.2	29.4	0.2	21.2	57.2	2.7
MEAN	**6.9 s**	**46.8 h**	**17.8**	**5 min 49.4 s**	**16.8**	**415.3**	**19.6**	**3.1 s**	**21.3**	**77.5**	**3.4**
adult	N/A			16839	72	7561	105.0	399.4	61	395	6.5
cid	N/A			---	---	---	---	5,938.0	15	95	6.3
forc	N/A			21542	63	2438	38.7	582.0	59	2105	35.7
mush	N/A			257.4	2	18	9.0	0.1	8	21	2.6
MEAN	N/A			---	---	---	---	**1729.9**	**35.8**	**654.0**	**12.8**

Next, DataSqueezer was analyzed in terms of the number of selectors, and the number of selectors per rule, which it generates. The number of selectors per rule generated by the algorithm for the 16 data sets is 3.4. Hence, on average, each rule generated by the DataSqueezer algorithm involves only 3.4 attribute-values pairs in the rule description. This is significantly less than the number of selectors per rule achieved by the CLIP4 algorithm (about 80% less). For comparison, the average number of attributes, without considering number of values they can take on, for the first 16 data sets is 17.3. Hence, the DataSqueezer uses on average only 20% of the attributes in the generated rules. Similarly, for the latter four data sets, DataSqueezer generates rules, which on average use only 40% of the attributes, since the average number of attributes for these data sets is 32.5. One can conclude that the rules generated by the DataSqueezer are very compact. The algorithm not only generates easy to understand format of generated knowledge, i.e., production rules, but also the rules are very short are, therefore, simple.

The DataSqueezer's average execution time for the first 16 data sets is 3.1 seconds. In (Lim et al., 2000) the authors reported the minimum execution time of 5 seconds for C4.5 algorithm (Quinlan, 1993; Quinlan 1996). The results show the DataSqueezer algorithm is much faster than any of the 33 ML algorithms. This shows that the algorithm generates the knowledge efficiently. It is also interesting to note that the POLYCLASS algorithm, which achieved the best accuracy, had an average execution time of 3.2h.

3.3.3.3 Flexibility

Based on the high validity (accuracy), simplicity and efficiency, which were achieved by the algorithm for the diverse range of the training sets, the algorithms is also highly flexible. It achieves very good performance for data sets that include both large number of examples with missing values (e.g., *adult*, *cid*, *mush* data sets), and large number of inconsistent, and thus noisy examples (e.g., *bld*, *cmc*, *led*, *pid*, *smo*, *tae*, and *adult* data sets). The algorithm also achieves very good performance for the data sets that incorporate both large number of missing and inconsistent examples (e.g., *adult* data set). We note, that the tests were performed for data sets that incorporate all three types of attributes: nominal, discrete numerical, and continuous numerical. Thus, the DataSqueezer algorithm is also characterized by very high flexibility.

3.3.3.4 Experimental Complexity Analysis

DataSqueezer was also experimentally tested to show additional validation of its linear complexity. The tests were performed using the *cid* data set, see Table 7. The data set includes 40 attributes and 300K of data samples, which assures the accuracy of the provided analysis. The training part of the original data set was used to derive training sets for the complexity analysis. The training sets were derived by selecting a number of examples from the beginning of the original training set. A standard procedure of using training sets of doubled size for each test, to verify if the execution time is also doubled, was used. The results are summarized in Table 10 (Kurgan and Cios, 2003a). The table shows the time ratios for the subsequent experiments, along with the results of the verification test, and the number of generated rules and selectors.

Table 10. Summary of experimental complexity analysis results for the DataSqueezer algorithm

Train data size	1000	2000	4000	8000	16000	32000	64000	128000	199523
Train data size (ratio)	---	2	2	2	2	2	2	2	1.56
Time	30msec	58msec	98msec	2sec 47msec	6sec 90msec	18sec 89msec	47sec 13msec	2min 16sec 29msec	4min 25sec 09msec
Time - Ratio	---	1.93	1.69	2.52	2.79	2.74	2.50	2.95	1.94
Accuracy	87.9	89.9	88.1	87.7	87.9	89.3	89.4	89.1	90.5
Sensitivity	90.3	92.3	90.7	90.3	90.5	92.1	92.2	91.8	93.5
Specificity	52.0	45.6	48.7	49.1	48.4	46.5	46.4	47.3	45.4
# Rules	11	13	10	12	12	13	14	13	15
# Selectors	74	80	57	77	74	81	88	83	95

Figure 12 visualizes the results. The figure shows a graph of relation between execution time and size of the data. It uses logarithmic scale on both axes to help visualize data points for low values.

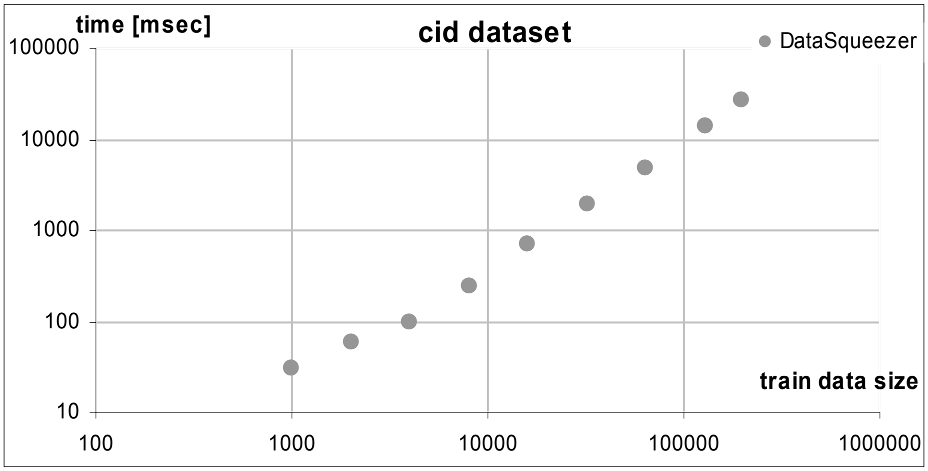


Figure 12. Relation between execution time and input data size for the DataSqueezer algorithm

The experimental results show linear relationship between the execution time and size of the input data for both algorithms, and thus agree with the theoretical complexity analysis. The time ratio is always close to the data size ratio, which implies that the DataSqueezer algorithm has linear complexity.

3.3.3.5 Summary of Experimental Evaluation

To summarize, two main advantages of the DataSqueezer algorithm are the compactness of the generated rules and low computational cost. The experimental results show that the algorithm strongly exhibits all four qualities of supervised inductive ML algorithms. It efficiently generates very simple rules that perform with high accuracy on test sets. It is also flexible to handle all types of attributes, and training sets that incorporate high number of noisy and missing values examples. Table 11 summarizes the results of the experimental evaluation of the DataSqueezer algorithm.

Table 11. Summary of the benchmarking tests for the DataSqueezer algorithm

accuracy	simplicity	efficiency	flexibility
high	high	very high	highly flexible: handles all attribute types noise resistant missing values resistant

The results place the DataSqueezer algorithm among the best ML algorithms. The reader is encouraged to examine Appendix C for detailed results, which include graphs and detailed reports from 10 fold cross validation runs.

3.4 The CAIM Algorithm

Since the DataSqueezer algorithm handles only numerical or nominal data, the CAIM discretization algorithm is used as a front-end to handle continuous attributes (Kurgan and Cios, 2001; Kurgan and Cios, 2002b; Kurgan and Cios, 2003b). There are many other inductive ML algorithms, e.g., AQ algorithms, CLIP algorithms, and CN2 algorithm that can handle only numerical or nominal data. It was also observed that some other ML algorithms that handle continuous attributes still perform better with discrete-valued attributes (Catlett, 1991; Kerber, 1992).

3.4.1 Introduction to Discretization

Discretization transforms a continuous attribute's values into a finite number of intervals and associates with each interval a numerical, discrete value. For mixed-mode (continuous and discrete) data, discretization is usually performed prior to the learning process (Catlett, 1991; Dougherty et al., 1995; Fayyad and Irani, 1992; Pfahringer, 1995). Discretization can be broken into two tasks. The first task is to find the number of discrete intervals. Only some discretization algorithms perform this; often the user must specify the number of intervals, or provide a heuristic rule (Ching et al., 1995). The second task is to find the width or the boundaries for the intervals, given the range of values of a continuous attribute. The CAIM algorithm automatically selects the number of discrete intervals and, at the same time, finds the width of every interval based on the interdependency between the classes and attribute values.

Discretization algorithms can be divided into two categories:

- unsupervised algorithms that discretize attributes without taking into account respective class labels. The two representative algorithms are equal-width and equal-frequency discretizations (Chiu, 1991). The equal-width discretization algorithm determines the minimum and maximum values of the discretized attribute, and then divides the range into a user-defined number of equal width discrete intervals. The equal-frequency algorithm determines the minimum and maximum values of the discretized attribute, sorts all values in ascending order, and divides the range into a user-defined number of intervals so that every interval contains the same number of sorted values.

- supervised algorithms discretize attributes by taking into account the interdependence between class labels and the attribute values. The representative algorithms are: maximum entropy (Wong and Chiu, 1987), Patterson-Niblett (Patterson and Niblett, 1987), which is built into a decision trees algorithm (Quinlan, 1993), Information Entropy Maximization (IEM) (Fayyad and Irani, 1993), and other information-gain or entropy-based algorithms (Dougherty et al., 1995; Wu, 1996), statistics-based algorithms, e.g., ChiMerge (Kerber, 1992) and Chi2 (Liu and Setiono, 1997), class-attribute interdependency algorithms, e.g., CADD (Ching et al., 1995), and clustering-based algorithms, e.g., K-means discretization (Tou and Gonzalez, 1974).

In addition, quantization methods (Linde et al., 1980) are also used to design discretization algorithms, e.g., the adaptive quantizer algorithm (Chan, 1991). Since large numbers of possible attribute values slows down and makes inductive learning ineffective, one of the main goals of a discretization algorithm is to significantly reduce the number of discrete intervals derived for a continuous attribute (Catlett, 1991). At the same time the algorithm should maximize the interdependency between discrete attribute values and class labels, as this minimizes the information loss due to discretization. As always, a satisfactory trade-off between these two goals needs to be achieved.

The CAIM algorithm discretizes an attribute into the smallest number of intervals and maximizes the class-attribute interdependency, and thus makes the subsequently performed ML task much easier. The algorithm automatically selects the number of discrete intervals without any user supervision. It uses class-attribute interdependency as defined in (Ching et al., 1995).

The CAIM algorithm was compared with six well-known discretization algorithms, almost always resulting in the smallest number of discrete intervals and the highest class-attribute interdependency. The CAIM algorithm and the six algorithms were used to discretize several continuous and mixed-mode data sets. The data sets were used with two ML algorithms - the CLIP4 (Cios and Kurgan, 2001; Cios and Kurgan 2002a), and C5.0 (Data Mining Tools, 2002), algorithms - to generate the rules. The accuracy of the rules shows that the application of the CAIM algorithm as a front-end discretization algorithm significantly improves performance of classification and also reduces the number of generated rules.

3.4.2 The Algorithm

Before describing the CAIM algorithm, the necessary background information is given. First, class-attribute interdependent discretization is described. Next, the CAIM discretization criterion, which is the core element of the CAIM algorithm, is described.

3.4.2.1 Definitions of Class-Attribute Interdependent Discretization

The CAIM algorithm's goal is to find the minimum number of discrete intervals while minimizing the loss of class-attribute interdependency. The algorithm uses class-attribute interdependency information as the criterion for the optimal discretization. Next, several definitions to define the criterion are introduced (Ching, 1995).

A supervised classification task requires a training data set consisting of *S* examples, where each example belongs to only one of *C* classes. Let *F* indicate any of the continuous attributes from the mixed-mode data. Then there exists a discretization scheme *D* on *F*, which discretizes the continuous domain of attribute *F* into *n* discrete intervals bounded by the pairs of numbers:

$$\mathrm{D}:\{[\mathrm{d}_0,\mathrm{d}_1],(\mathrm{d}_1,\mathrm{d}_2],\ldots,(\mathrm{d}_{\mathrm{n}\text{-}1},\mathrm{d}_\mathrm{n}]\}$$

where d_0 is the minimal value and d_n is the maximal value of attribute *F*, and the values are arranged in ascending order. These values constitute the boundary set $\{d_0, d_1, d_2, \ldots, d_{n-1}, d_n\}$ for discretization *D*.

Each value of attribute *F* can be classified into only one of the *n* intervals defined above. Membership of each value within a certain interval for attribute *F* may change with the change of the discretization *D*. The class variable and the discretization variable of attribute *F* are treated as two random variables defining a two-dimensional frequency matrix (called quanta matrix) that is shown in Table 12.

Table 12. 2-D quanta matrix for attribute F and discretization scheme D

Class	Interval					Class Total
	$[d_0, d_1]$	…	$(d_{r-1}, d_r]$	…	$(d_{n-1}, d_n]$	
C_1	q_{11}	…	q_{1r}	…	q_{1n}	M_{1+}
:	:	…	:	…	:	:
C_i	q_{i1}	…	q_{ir}	…	q_{in}	M_{i+}
:	:	…	:	…	:	:
C_C	q_{C1}	…	q_{Cr}	…	q_{Cn}	M_{C+}
Interval Total	M_{+1}	…	M_{+r}	…	M_{+n}	M

In Table 12, q_{ir} is the total number of continuous values belonging to the ith class that are within interval $(d_{r-1}, d_r]$, M_{i+} is the total number of objects belonging to the ith class, and M_{+r} is the total number of continuous values of attribute F that are within the interval $(d_{r-1}, d_r]$, for i=1,2…,C and, r= 1,2, …, n.

The estimated joint probability of the occurrence that attribute F values are within the interval $D_r = (d_{r-1}, d_r]$, and belong to class C_i can be calculated as:

$$p_{ir} = p(C_i, D_r \mid F) = \frac{q_{ir}}{M}.$$

The estimated class marginal probability that attribute F values belong to class C_i, p_{i+}, and the estimated interval marginal probability that attribute F values are within the interval $D_r = (d_{r-1}, d_r]$ p_{+r} are as follows:

$$p_{i+} = p(C_i) = \frac{M_{i+}}{M} \text{ and } p_{+r} = p(D_r \mid F) = \frac{M_{+r}}{M}.$$

The Class-Attribute Mutual Information between the class variable C and the discretization variable D for attribute F given the 2-D frequency matrix shown in Table 12 is defined as:

$$I(C,D\,|\,F)=\sum_{i=1}^{C}\sum_{r=1}^{n}p_{ir}\log_2\frac{p_{ir}}{p_{i+}p_{+r}}.$$

Similarly, the Class-Attribute Information (Fayyad and Irani, 1992) and the Shannon's entropy of the given matrix are defined, respectively, as:

$$INFO(C,D\,|\,F)=\sum_{i=1}^{C}\sum_{r=1}^{n}p_{ir}\log_2\frac{p_{+r}}{p_{ir}} \text{ and } H(C,D\,|\,F)=\sum_{i=1}^{C}\sum_{r=1}^{n}p_{ir}\log_2\frac{1}{p_{ir}}.$$

Given the three latter equations, the Class-Attribute Interdependence Redundancy (CAIR) criterion (Wong and Liu, 1975) and Class-Attribute Interdependence Uncertainty (CAIU) (Huang, 1996) criterion are defined as follows:

$$R(C,D\,|\,F)=\frac{I(C,D\,|\,F)}{H(C,D\,|\,F)} \text{ and } U(C,D\,|\,F)=\frac{INFO(C,D\,|\,F)}{H(C,D\,|\,F)}.$$

The CAIR criterion is used in the Class-Attribute Dependent Discretizer (CADD) algorithm (Ching, 1995). The CAIR criterion is used to measure the interdependence between classes and the discretized attribute (the larger its value the better correlated are the class labels and the discrete intervals) (Cios et al, 1998). It is also independent of the number of class labels and the number of unique values of the continuous attribute. The same holds true for the CAIU criterion, but with a reverse relationship. The CADD algorithm has the following disadvantages:

- it uses a user-specified number of intervals when initializing the discretization intervals,

- it initializes the discretization intervals using a maximum entropy discretization method; such initialization may be the worst starting point in terms of the CAIR criterion,
- the significance test used in the algorithm requires training for selection of a confidence interval.

The CAIU and CAIR criteria were both used in the CAIUR discretization algorithm (Huang, 1996). The CAIUR algorithm avoided the disadvantages of the CADD algorithm generating discretization schemes with higher CAIR values, but at the expense of a very high computational cost, making it inapplicable for discretization of continuous attributes that have a large number of unique values.

The CAIM algorithm has the following three goals:

- to maximize the interdependency between the continuous-valued attribute and its class labels,
- to achieve the minimum number of discrete intervals possible,
- to perform the discretization task at reasonable computational cost so that it can be applied to continuous attributes with large number of unique values.

The CAIM algorithm avoids the disadvantages of the CADD and CAIUR algorithms. It works in a top-down manner, dividing one of the existing intervals into two new intervals using a criterion that results in achieving the optimal class-attribute interdependency after the split, and starts with a single, [d_o, d_n], interval.

3.4.2.2 Discretization Criterion

The Class-Attribute Interdependency Maximization (CAIM) criterion measures the dependency between the class variable C and the discretization variable D for attribute F, for a given quanta matrix (see Table 12), and is defined as:

$$CAIM(C,D\,|\,F) = \frac{\sum_{r=1}^{n} \frac{\max_r{}^2}{M_{+r}}}{n},$$

where: n is the number of intervals, r iterates through all intervals, i.e., r=1,2,...,n, max_r is the maximum value among all q_{ir} values (maximum value within the r^{th} column of the quanta matrix), i=1,2,...,C, M_{+r} is the total number of continuous values of attribute F that are within the interval (d_{r-1}, d_r].

The CAIM criterion is a heuristic measure that is used to quantify the interdependence between classes and the discretized attribute. It has the following properties:

- the larger the value of CAIM criterion the higher the interdependence between the class labels and the discrete intervals. The bigger the number of values belonging to class C_i within a particular interval (if the number of values belonging to C_i within the interval is the largest then C_i is called the leading class within the interval) the higher the interdependence between C_i and the interval. The goal of maximizing the interdependence between classes and the discrete intervals can be translated into achieving the largest possible number of values that belong to a leading class within all intervals. The CAIM criterion accounts for the trend of maximizing the number of values belonging to a leading class within each interval by using max_i. The value of CAIM criterion grows when values of max_r grow, which relates to the increase of the interdependence between the class labels and the discrete intervals. The highest interdependence between the class labels and the discrete intervals (and at the same time the highest value of CAIM) is achieved when all values within a particular interval belong to the same class for all intervals. In this case, $max_r = M_{+i}$ and CAIM=S/n,

- it takes on real values from the interval [0, S] where S is the number of values of the continuous attribute F,
- the criterion generates discretization schemes where each interval has all of its values grouped within a single class label. This observation motivated us to use the max_r values within each of the n intervals, and summing them for all intervals,
- the squared max_i value is divided by the M_{+i} for two reasons:
  - to account for the negative impact that values belonging to classes other than the class with the maximum number of values within an

interval have on the discretization scheme. The more such values the bigger the value of M_{+i}, which in turn decreases the value of CAIM.

- to scale the max_r^2 number. Because the division factor M_{+i} is always greater than or equal to max_r, the overflow error will not happen during calculations. To avoid the overflow, the calculation is performed by first dividing max_r by M_{+i} and then multiplying the result by max_r, i.e.,

  $$\frac{\max_r^{\,2}}{M_{+i}} \; \textit{is calculated as} \; \frac{\max_r}{M_{+i}}\max_r,$$

- because the algorithm favors discretization schemes with smaller numbers of intervals, the summed value is divided by the number of intervals n,
- the M_{i+} values from the quanta matrix are not used because they are defined as the total number of objects belonging to the i^th class, which does not change with different discretization schemes.

The value of the CAIM criterion is calculated with a single pass over the quanta matrix. The CAIM criterion has similar properties to the CAIR criterion but the experimental results show that the CAIM criterion tends to generate a much smaller number of intervals and using it results in achieving higher interdependency. The CAIM criterion is used by the CAIM algorithm to perform discretization.

3.4.2.3 The CAIM Algorithm

The optimal discretization scheme can be found by searching over the space of all possible discretization schemes to find the one with the highest value of the CAIM criterion. Such a search for a scheme with the globally optimal value of CAIM is highly combinatorial and time consuming. Thus, the CAIM algorithm uses a greedy approach, which searches for the approximate optimal value of the CAIM criterion by finding local maximum values of the criterion. Although this approach does not guarantee finding the global maximum, it is both computationally inexpensive and closely-approximates the optimal discretization scheme, which is shown in section 3.4.4. The algorithm consists of these two steps:

- initialization of the candidate interval boundaries and the initial discretization scheme,
- consecutive additions of a new boundary that results in the locally highest value of the CAIM criterion.

The pseudocode of the CAIM algorithm is given in Figure 13.

The algorithm starts with a single interval that covers all possible values of a continuous attribute, and divides it iteratively. From all possible division points that are tried (with replacement) in 2.2., it chooses the division boundary that gives the highest value of the CAIM criterion. The algorithm assumes that every discretized attribute needs at least the number of intervals equal to the number of classes since this assures us that the discretized attribute can improve subsequent classification.

```
Given: Data consisting of S examples, C classes, and continuous attributes $F_i$
For every $F_i$ do:
Step1.
1.1  find maximum ($d_n$) and minimum ($d_o$) values of $F_i$
1.2  form a set of all distinct values of $F_i$ in ascending order, and initialize all possible interval boundaries B
     with minimum, maximum and all the midpoints of all the adjacent pairs in the set
1.3  set the initial discretization scheme as D:{[$d_0$, $d_n$]}, set GlobalCAIM=0
Step2.
2.1  initialize k=1;
2.2  tentatively add an inner boundary, which is not already in D, from B, and calculate corresponding CAIM
     value
2.3  after all the tentative additions have been tried accept the one with the highest value of CAIM
2.4  if (CAIM > GlobalCAIM or k<C) then update D with the accepted in step 2.3 boundary and set
     GlobalCAIM=CAIM, else terminate
2.5  set k=k+1 and go to 2.2
Output: Discretization scheme D
```

Figure 13. The pseudo-code of the CAIM algorithm

The CAIM algorithm implements a balance between a reasonable computational cost and finding the optimal discretization scheme. Despite the greedy manner in which the algorithm works, the discretization schemes it generates have very high class-attribute interdependency and always a small number of discretization intervals. For the data sets used in the experimental section, the CAIM algorithm generated discretization schemes with the smallest number of intervals that assures low computational cost, and always achieved very high class-attribute interdependency, which results in significant improvement in the subsequently performed classification tasks.

3.4.2.4 Complexity Analysis

In what follows we determine the complexity of the algorithm for discretizing a single attribute. The CAIM algorithm's time bound is determined by the calculation of the CAIM criterion in step 2.2. In the worst case, the CAIM criterion is calculated in O(C·S) time, where S is the number of distinct values of the discretized attribute, C is the number of classes in the problem, and usually is a small constant. The CAIM algorithm starts with a single interval, and as experimental results show, the expected number of intervals per attribute is O(C). Thus, the time bound for calculation of the CAIM value can be estimated as O(C^2). The CAIM values are calculated in O(S) time for all candidate boundaries in step 2.2. This gives the total time of step 2.2 as O(S·C^2). Step 2.2 is executed in the worst case in O(S), and the results show that the expected number of intervals is again O(C), thus we can estimate that step 2.2 is executed in O(C). Therefore, the time bound for Step 2 of the CAIM algorithm is O(C)·O(S·C^2) = O(S·C^3). Sorting in step 1.2 takes O(S·log S) time, and determines the time for Step1. Depending on the value of C, which for most inductive machine learning applications is a small constant, the expected running time of the algorithm is O(S·log S). This shows that the CAIM algorithm can be applied to large problems.

The remaining costs of the algorithm include building the quanta matrix given the discretization scheme in O(S) time (this time adds to calculating the CAIM value), updating the discretization scheme in step 2.4 in O(S) time, and updating the global CAIU value in O(C) time. All these costs are negligible.

3.4.3 Experimental Evaluation

Below, the results of the CAIM algorithm along with the other six leading discretization algorithms on the eight well-known continuous and mixed-mode data sets are presented. The *smo* data, see Table 13, set was obtained from the StatLib project data sets repository (Vlachos, 2000), and remaining data sets were obtained from the University of California Irvine ML Repository (Blake and Merz, 1998). Detailed description of the data sets is shown in Table 13.

Table 13. Description of data sets used for benchmarking of CAIM algorithm

Properties	Data sets							
	iris	sat	thy	wav	ion	smo	hea	pid
# of classes	3	6	3	3	2	3	2	2
# of examples	150	6435	7200	3600	351	2855	270	768
# of training / testing examples	10 x cross-validation	10 x cross-validation	10 x cross-validation	10 x cross-validation	10 x cross-validation	10 x cross-validation	10 x cross-validation	10 x cross-validation
# of attributes	4	36	21	21	34	13	13	8
# of continuous attributes	4	36	6	21	32	2	6	8

The benchmarking tests are characterized by diversity of training sets, including their size, both in terms of number of examples and attributes, and number of classes. The range of the tests is characterized by:

- the size of training data sets: between 150 and 7200 examples,
- the number of attributes: between 4 and 36,
- the number of classes: between 2 and 6.

This diversity of the tests ensured that evaluation of the algorithm, which is performed based on comparison with results achieved by other state of the art discretization algorithms, is comprehensive and strong.

3.4.4 Comparison of CAIM with other Algorithms

Tests were performed for the CAIM algorithm and six other discretization algorithms. The six discretization algorithms were:

- two unsupervised algorithms: equal-width and equal frequency,
- four supervised algorithms: Patterson-Niblett, IEM, Maximum Entropy and CADD.

The unsupervised algorithms require the user to specify the number of discrete intervals. In our experiments we used the following heuristic formula to estimate the number of intervals: $n_{Fi} = M / (3C)$, where n_{Fi} is the number of intervals for attribute Fi, M is the number of examples, and C is the number of classes (Wong and Chiu, 1987). The supervised algorithms apply their own criteria to generate an appropriate number of discrete intervals.

All seven algorithms were used to discretize the eight data sets. The goodness of the discretization algorithm was evaluated based on the CAIR criterion value, the number of generated intervals, and the execution time.

To quantify the impact of the selection of a discretization algorithm on the classification task performed subsequently by a ML algorithm, the discretized data sets were used to generate rules by ML algorithms. The CLIP4 algorithm was used to represent the hybrid algorithms, and the C5.0 algorithm to represent decision tree algorithms. The classification goodness was measured using accuracy and the number of rules. The results were compared among the seven discretization algorithms, for all data sets and both learning algorithms.

3.4.4.1 Accuracy, Simplicity, Efficiency, and Flexibility

Evaluation of the discretization algorithms was performed using the CAIR criterion since one of the goals of discretization is to maximize the class-attribute interdependence (Wong and Liu, 1975). This can be done by finding a discretization scheme, D_{MAX}, out of all possible discretization schemes, D, such that: $\text{CAIR}(D_{MAX}) \geq \text{CAIR}(D_i)\ \forall(D_i \in D)$ (Ching et al. 1995).

The CAIM criterion has the same properties as the CAIR criterion, but since it is a new heuristic measure, the CAIR criterion was used instead. The higher the value of the CAIR criterion, the higher the interdependence between the class labels and the discrete intervals. Table 14 shows the CAIR value, the number of discrete intervals, and the execution time for the 10-fold cross validation tests on 8 data sets, and the seven discretization schemes (Kurgan and Cios, 2002b). The discretization was done using the training folds, and the testing folds were discretized using the already generated discretization scheme. The direct comparison of results can be performed by looking at the rank column in Table 14. The rank value is defined as each algorithm's rank for a particular data set among the seven algorithms, averaged over the eight data sets.

The CAIM algorithm achieved the highest class-attribute interdependency for 5 out of 8 data sets, and for *wav* and *ion* data sets had the second and third highest, respectively. The CAIM algorithm was behind the competitors for only the *smo* data set, but this data set has only 2 continuous attributes out of 13. For this test, the CAIM algorithm achieved the highest rank (1.9) among all compared algorithms, and this rank is significantly better than 3.1 achieved by the Information Entropy Maximization algorithm, which was the second best. The results show that the greedy approach combined with the CAIM criterion work in practice resulting, on average, in higher interdependence between class and attribute variables than the interdependence achieved by other algorithms. This, in turn, implies that the CAIM algorithm is characterized by high accuracy.

Table 14. Comparison of the seven discretization algorithms using eight continuous and mixed-mode data sets (bolded values indicate the best results)

Criterion	**Discretization Method**	**Data set**																**RANK**
		iris	std	**sat**	std	**thy**	std	**wav**	std	**ion**	std	**smo**	std	**hea**	std	**pid**	std	mean
CAIR mean value through all intervals	Equal Width	0.40	0.01	0.24	0	0.071	0	0.068	0	0.098	0	0.011	0	0.087	0	0.058	0	4.0
	Equal Frequency	0.41	0.01	0.24	0	0.038	0	0.064	0	0.095	0	0.010	0	0.079	0	0.052	0	5.4
	Paterson-Niblett	0.35	0.01	0.21	0	0.144	0.01	**0.141**	0	0.192	0	0.012	0	0.088	0	0.052	0	3.5
	Maximum Entropy	0.30	0.01	0.21	0	0.032	0	0.062	0	0.100	0	0.011	0	0.081	0	0.048	0	5.9
	CADD	0.51	0.01	**0.26**	0	0.026	0	0.068	0	0.130	0	**0.015**	0	0.098	0.01	0.057	0	3.4
	IEM	0.52	0.01	0.22	0	0.141	0.01	0.112	0	**0.193**	0.01	0.000	0	0.118	0.02	0.079	0.01	3.1
	CAIM	**0.54**	0.01	**0.26**	0	**0.170**	0.01	0.130	0	0.168	0	0.010	0	**0.138**	0.01	**0.084**	0	**1.9**
total # of intervals	Equal Width	16	0	252	0	126	0.48	630	0	640	0	22	0.48	56	0	106	0	4.8
	Equal Frequency	16	0	252	0	126	0.48	630	0	640	0	22	0.48	56	0	106	0	4.8
	Paterson-Niblett	48	0	432	0	45	0.79	252	0	384	0	17	0.52	48	0.53	62	0.48	4.0
	Maximum Entropy	16	0	252	0	125	0.52	630	0	572	6.70	22	0.48	56	0.42	97	0.32	4.4
	CADD	16	0.71	246	1.26	84	3.48	628	1.43	536	10.26	22	0.48	55	0.32	96	0.92	3.6
	IEM	**12**	0.48	430	4.88	28	1.60	91	1.50	113	17.69	**2**	0	**10**	0.48	17	1.27	2.1
	CAIM	**12**	0	**216**	0	**18**	0	**63**	0	**64**	0	6	0	12	0	**16**	0	**1.3**
time [s]	Equal Width	**0.02**	0.01	26.63	2.02	**4.74**	0.05	**8.04**	0.26	**1.21**	0.02	0.32	0.01	**0.08**	0.01	**0.27**	0.01	**1.3**
	Equal Frequency	0.03	0.01	**26.11**	0.55	4.85	0.16	8.46	0.21	1.29	0.06	**0.31**	0.01	**0.08**	0	**0.27**	0.01	1.5
	Paterson-Niblett	0.12	0.01	82.52	2.36	32.98	1.60	176.8	4.13	14.35	2.75	1.44	0.06	0.42	0.01	2.44	0.01	6.4
	Maximum Entropy	0.03	0	30.36	1.42	11.01	0.65	26.76	1.66	3.33	0.39	0.45	0.05	0.14	0.02	0.58	0.05	3.5
	CADD	0.08	0.01	54.19	1.81	65.46	13.62	435.5	24.68	17.88	0.65	1.05	0.16	0.69	0.05	3.91	0.30	6.6
	IEM	0.05	0	49.90	1.77	10.56	0.35	59.14	3.16	2.57	0.32	0.53	0.01	0.14	0.01	0.77	0.05	4.1
	CAIM	0.05	0.01	53.36	1.90	11.50	0.47	46.13	3.68	2.51	0.25	0.64	0.01	0.13	0.01	0.70	0.01	4.1

The CAIM algorithm generated discretization scheme with the smallest number of intervals for 6 data sets, as compared with six other discretization algorithms. For the *smo* and *hea* data sets, it generated the second smallest number of intervals. Again, the rank of CAIM was significantly better than the ranks of other discretization algorithms. Smaller numbers of discrete intervals reduces the size of the data and helps to better understand the meaning of the discretized attributes. This is a significant advantage of the CAIM algorithm that further shows its usefulness and proves that the algorithm is characterized by high simplicity.

Unsupervised discretization algorithms achieved the shortest execution time since they do not process any class related information; they require less computation time and generate results that are less suited for the subsequent ML tasks. Among supervised algorithms, the Maximum Entropy algorithm achieved the best average rank. The second fastest were IEM and CAIM algorithms; they worked well on larger data sets like *thy* or *wav*, which is important for real-life applications. The results for IEM, CAIM, and Maximum Entropy algorithms show that they are the most efficient among supervised methods, with comparable performance.

The above results show good applicability of the CAIM algorithm, which generates small numbers of intervals that are highly interdependent with class labels, with speeds comparable to the fastest supervised discretization algorithms.

Based on the accuracy, simplicity, and efficiency achieved on diverse training datasets, the algorithm was shown to be highly flexible.

3.4.4.2 Impact of the CAIM algorithm discretization on the Subsequent Learning Task

The discretized data sets were used as input to CLIP4 and C5.0 algorithms to generate rules. The accuracy and the number of rules were compared for the seven discretization algorithms. Since C5.0 can generate data models from continuous attributes, its performance while generating rules from raw data, against the results achieved using discretized data using the seven algorithms was also compared. Direct comparison of results can be seen by looking at the RANK column in Table 15 that shows the accuracy (Kurgan and Cios, 2002b).

On average, the best accuracy for the two inductive ML algorithms was achieved for the data that was discretized using the CAIM algorithm. Using CLIP4 and C5.0 to generate a data model, the difference between the rank achieved by the CAIM algorithm and the next best IEM algorithm, and built-in discretization, in the case of C5.0, is over 1.0. In the case of using the CLIP4 algorithm to generate a data model, the average accuracy of the rules was the highest for data discretized with the CAIM algorithm. The second best accuracy was achieved for the data discretized with the IEM algorithm, while accuracies using the remaining discretization algorithms were lower and comparable to each other.

Table 15. Comparison of the accuracies achieved by the CLIP4 and C5.0 algorithms for the eight data sets using the seven discretization schemes (bolded values indicate the best results)

Algor.	**Discretization Method**	**Data sets**																**RANK**
		iris		**Sat**		**thy**		**wav**		**ion**		**smo**		**hea**		**pid**		mean
		acc	std	acc	std	acc	std	acc	std	acc	std	acc	std	acc	std	acc	std	
CLIP4 accuracy	Equal Width	88.0	6.9	**77.5**	2.8	91.7	1.9	68.2	2.2	86.9	6.4	68.6	2.0	64.5	10.1	65.5	6.5	4.6
	Equal Frequency	91.2	7.6	76.3	3.4	95.7	2.4	65.4	2.9	81.0	3.7	68.9	2.8	72.6	8.2	63.3	6.3	4.8
	Paterson-Niblett	87.3	8.6	75.6	3.9	97.4	0.6	60.9	5.2	**93.7**	3.9	68.9	2.7	68.5	13.6	72.7	5.1	4.3
	Maximum Entropy	90.0	6.5	76.4	2.7	97.3	0.9	63.5	2.9	82.9	4.8	68.7	2.8	62.6	9.8	63.4	5.1	5.3
	CADD	**93.3**	4.4	**77.5**	2.6	70.1	13.9	61.5	3.4	88.8	3.1	68.8	2.5	72.2	11.4	65.5	4.2	3.9
	IEM	92.7	4.9	77.2	2.7	**98.8**	0.5	75.2	1.7	92.4	6.9	66.9	2.6	75.2	8.6	72.2	4.2	2.9
	CAIM	92.7	8.0	76.4	2.0	97.9	0.4	**76.0**	1.9	92.7	3.9	**69.8**	4.0	**79.3**	5.0	**72.9**	3.7	**1.8**
C5.0 accuracy	Equal Width	94.7	5.3	86.0	1.6	95.0	1.1	57.7	8.2	85.5	6.4	69.2	5.4	74.7	5.2	70.8	2.8	5.3
	Equal Frequency	94.0	5.8	85.1	1.5	97.6	1.2	57.5	7.9	81.0	12.4	70.1	1.7	69.3	5.7	70.3	5.4	6.0
	Paterson-Niblett	94.0	4.9	83.0	1.0	97.8	0.4	74.8	5.6	85.0	8.1	70.1	3.2	**79.9**	7.1	71.7	4.4	4.3
	Maximum Entropy	93.3	6.3	85.2	1.5	97.7	0.6	55.5	6.2	86.5	8.8	70.2	3.9	73.3	7.6	66.4	5.9	5.6
	CADD	93.3	5.4	86.1	0.9	93.5	0.8	56.9	2.1	77.5	11.9	70.2	4.7	73.6	10.6	71.8	2.2	5.4
	IEM	**95.3**	4.5	84.6	1.1	99.4	0.2	**76.6**	2.1	**92.6**	2.9	69.7	1.6	73.4	8.9	**75.8**	4.3	3.3
	CAIM	**95.3**	4.5	86.2	1.7	98.9	0.4	72.7	4.2	89.0	5.2	**70.3**	2.9	76.3	8.9	74.6	4.0	**2.1**
	Built-in	92.7	9.4	**86.4**	1.7	**99.8**	0.4	72.6	3.6	87.0	9.5	70.1	1.3	76.8	9.9	73.7	4.9	3.3

The average accuracy of rules generated by the C5.0 algorithm shows that the best results are achieved after discretization of data with the CAIM algorithm. The second best results were achieved by discretizing data using the IEM algorithm and C5.0 with its built-in discretization. Discretization using the remaining algorithms resulted in achieving significantly worse accuracies on the average. The accuracy results show that the CAIM algorithm generates the discrete data that results in improved performance of subsequently used supervised inductive ML algorithms when compared to the data generated by the other discretization algorithms. Table 16 shows the classification results in terms of number of generated rules (Kurgan and Cios, 2002b).

Table 16. Comparison of the number of rules/leaves generated by the CLIP4 and C5.0 algorithms for the eight data sets using the seven discretization (bolded values indicate the best results)

Algor.	**Discretization Method**	**Data sets**																**RANK**
		iris		**sat**		**thy**		**wav**		**ion**		**smo**		**pid**		**hea**		mean
		#	std	#	std	#	std	#	std	#	std	#	std	#	std	#	std	
CLIP4 # rules	Equal Width	4.2	0.4	47.9	1.2	**7.0**	0.0	**14.0**	0.0	**1.1**	0.3	20.0	0.0	7.3	0.5	7.0	0.5	3.8
	Equal Frequency	4.9	0.6	47.4	0.8	**7.0**	0.0	**14.0**	0.0	1.9	0.3	19.9	0.3	7.2	0.4	6.1	0.7	3.5
	Paterson-Niblett	5.2	0.4	**42.7**	0.8	**7.0**	0.0	**14.0**	0.0	2.0	0.0	19.3	0.7	**1.4**	0.5	7.0	1.1	2.6
	Maximum Entropy	6.5	0.7	47.1	0.9	**7.0**	0.0	**14.0**	0.0	2.1	0.3	19.8	0.6	7.0	0.0	**6.0**	0.7	3.6
	CADD	4.4	0.7	45.9	1.5	**7.0**	0.0	**14.0**	0.0	2.0	0.0	20.0	0.0	7.1	0.3	6.8	0.6	3.5
	IEM	4.0	0.5	44.7	0.9	**7.0**	0.0	**14.0**	0.0	2.1	0.7	18.9	0.6	3.6	0.5	8.3	0.5	3.0
	CAIM	**3.6**	0.5	45.6	0.7	**7.0**	0.0	**14.0**	0.0	1.9	0.3	**18.5**	0.5	1.9	0.3	7.6	0.5	**2.1**
C5.0 # rules	Equal Width	6.0	0.0	348.5	18.1	31.8	2.5	69.8	20.3	32.7	2.9	**1.0**	0.0	249.7	11.4	66.9	5.6	4.9
	Equal Frequency	4.2	0.6	367.0	14.1	56.4	4.8	56.3	10.6	36.5	6.5	**1.0**	0.0	303.4	7.8	82.3	0.6	5.8
	Paterson-Niblett	11.8	0.4	**243.4**	7.8	15.9	2.3	**41.3**	8.1	18.2	2.1	**1.0**	0.0	58.6	3.5	58.0	3.5	3.3
	Maximum Entropy	6.0	0.0	390.7	21.9	42.0	0.8	63.1	8.5	32.6	2.4	**1.0**	0.0	306.5	11.6	70.8	8.6	5.8
	CADD	4.0	0.0	346.6	12.0	35.7	2.9	72.5	15.7	24.6	5.1	**1.0**	0.0	249.7	15.9	73.2	5.8	4.9
	IEM	**3.2**	0.6	466.9	22.0	34.1	3.0	270.1	19.0	12.9	3.0	**1.0**	0.0	**11.5**	2.4	**16.2**	2.0	3.5
	CAIM	**3.2**	0.6	332.2	16.1	**10.9**	1.4	58.2	5.6	**7.7**	1.3	**1.0**	0.0	20.0	2.4	31.8	2.9	**1.9**
	Built-in	3.8	0.4	287.7	16.6	11.2	1.3	46.2	4.1	11.1	2.0	1.4	1.3	35.0	9.3	33.3	2.5	3.1

The rank achieved by the CAIM algorithm, for experiments performed with CLIP4 and C5.0 algorithms, shows that on average it had the smallest number of rules. Closer analysis of the results shows that the CLIP4 algorithm generates a small number of rules for all data sets discretized using the seven discretization algorithms. The average rank results show that discretizing data using Paterson-Niblett algorithm resulted in an average number of rules similar to the number of rules for data models generated using data discretized with the CAIM algorithm. On the other hand, the number of leaves (rules) generated by the C5.0 algorithm varied significantly over the data sets. The three discretization algorithms that work best with the C5.0 algorithm are: the CAIM algorithm, the Paterson-Niblett algorithm, and the IEM algorithm. Also, similarly low numbers of leaves were generated when using the C5.0's built-in discretization. Among these four discretization algorithms, discretizing the data using the CAIM algorithm resulted in the smallest average number of leaves.

The above tests show that using CAIM algorithm not only results in accurate, efficient, simple, and flexible discretization, but also results in significant improvement in accuracy and simplicity of results generated by subsequently applied inductive ML algorithms.

3.4.4.3 Summary

To summarize, the CAIM algorithm is a very efficient algorithm for discretization of continuous attributes. It generates very accurate and simple discretization schemes, when comparing to other state of the art discretization algorithms. It also improves accuracy and simplicity of results generated by inductive ML algorithms, when generating rules on discretized data. Table 17 summarizes the results of the experimental evaluation of the CAIM algorithm.

Table 17. Summary of the benchmarking tests for the CAIM algorithm

accuracy	simplicity	efficiency	flexibility
very high	very high	high	highly flexible improves accuracy and simplicity of subsequently used inductive ML algorithms

The results place the DataSqueezer algorithm among the best discretization algorithms.

The CAIM algorithm is used in the MetaSqueezer system because of its advantages shown above. Namely, it discretizes continuous attributes into (possibly) the smallest number of intervals, which results in better compactness of the discretized data. It maximizes class-attribute interdependency, which results in minimization of the information loss due to discretization, and improving results achieved by the subsequently performed inductive ML task. The CAIM algorithm also selects, on its own, the number of discrete intervals.

Chapter 4

4 The MetaSqueezer System

This chapter provides detailed description of the system. The description is followed by experimental and theoretical evaluation and comparison with other IL algorithms. In the Chapter 5 we will describe application of the system to analysis of cystic fibrosis data.

4.1 Introduction

The MetaSqueezer system is suitable to efficiently generate production rules for large quantities of supervised data (Kurgan and Cios, 2003a). The system uses MM concept for generation of rules. The rules are generated using a three-step process: preprocessing, DM, and MM. MetaSqueezer generates production rules by repeatedly applying the IL algorithm, DataSqueezer, which works based on the generalization operations, and irrevocable, informed hill climber search. The DataSqueezer algorithm is used in the DM step to generate meta-data, and again in the MM step to generate meta-rules, which constitute outcome from the MetaSqueezer system.

4.2 MetaSqueezer System

The pseudo-code of the MetaSqueezer system follows:

```
Given: supervised data organized into n training subsets, and describing c classes
    1.   use the DataSqueezer algorithm to generate rule sets, $RS_i$, i = 1,2…n, for all training subsets
    2.   for each $RS_i$ generate a rule table $RT_i$, i = 1,2…n
    3.   use the DataSqueezer algorithm to generate set of meta-rules from a data table being a concatenation
         of all $RT_i$
Result: the generated meta-rules
```

Figure 14. The pseudo-code of the CAIM algorithm

During the preprocessing step, the raw data are first preprocessed by performing data validation and transformation. The resulting data is divided into training subsets that are discretized using the CAIM algorithm, and then fed into the DM step. DM generates meta-data from the data subsets. The DataSqueezer algorithm is used to generate the meta-data, in terms of rules generated for each of the training subsets. Next, in the Meta Mining step the meta-data generated for each of the subsets is concatenated and fed again into the DataSqueezer algorithm to generate meta-rules. The detailed architecture of the MetaSqueezer system is shown in Figure 15.

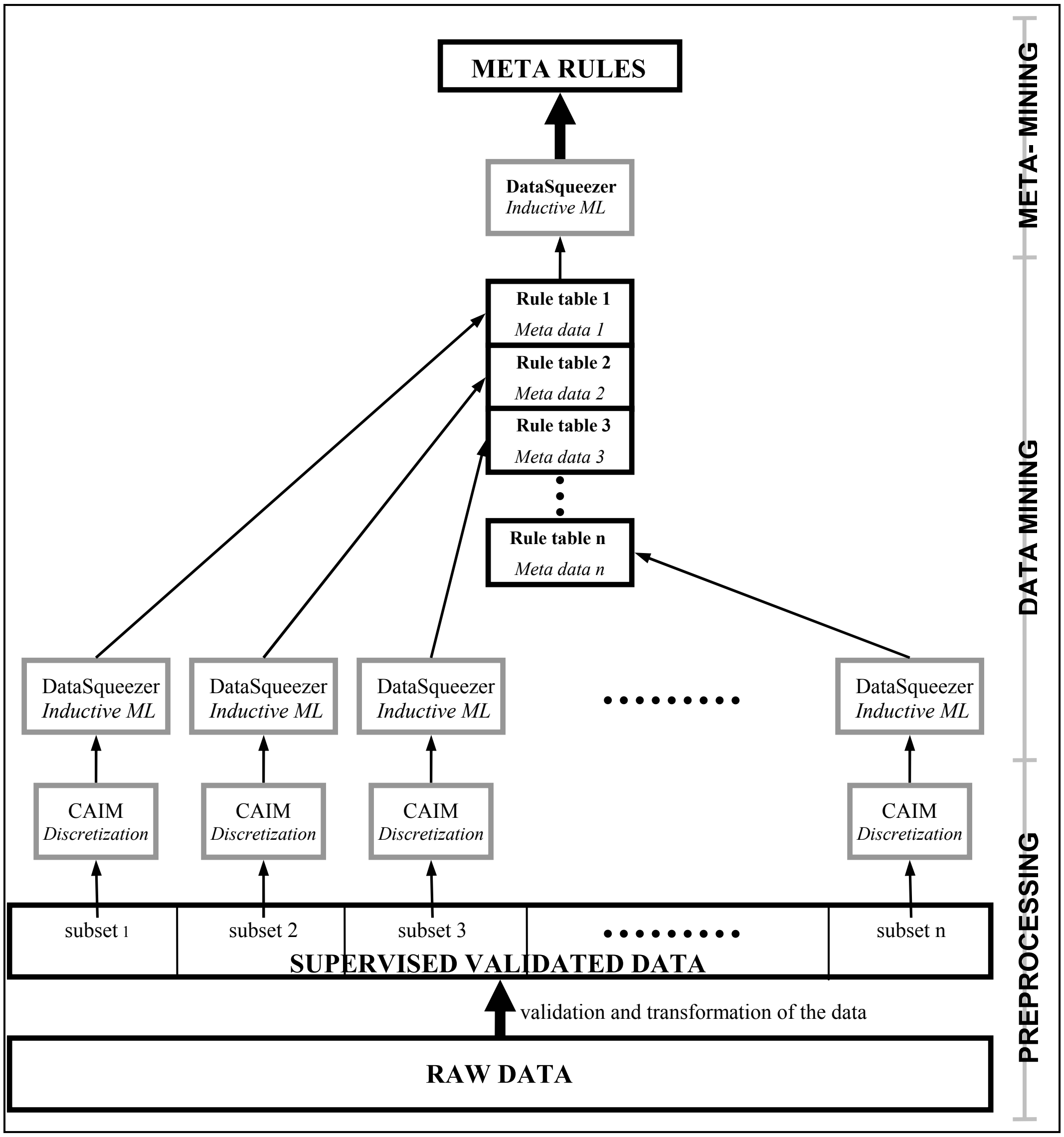


Figure 15. Architecture of the MetaSqueezer system

Since the MetaSqueezer applies the DataSqueezer algorithm to generate both meta-data and meta-rules, its properties are the same as the properties of the DataSqueezer algorithm, see Table 18.

Table 18. Major properties of the MetaSqueezer system

Search Process	Search Type	Results	Other Features
top-down and bottom-up	irrevocable hill climber	production meta-rules	all attribute types, noise and missing values resistant

The MetaSqueezer system has the following characteristic features:

- it generates production rules that involve no more than one selector per attribute. This is because the DataSqueezer algorithm, which is used to generate meta-rules, generates rules with a single selector per attribute.
- it generates rules that are very compact in terms of the number of used selectors. Experimental results, shown in section 4.3, indicate that the MetaSqueezer system generates rules that involve small number of selectors, even smaller than for the rules generated from the same training data by the DataSqueezer algorithm. The main reason for the compactness is application of the MM concept in the MetaSqueezer system. Since the outcome from the system, meta-rules, are generated from already generated meta-data, they provide information about patterns exhibited in the already summarized data.
- it can handle data with large number of missing values and large number of noisy examples, since it applies the DataSqueezer algorithm that is highly noise and missing values resistant.
- it generates independent rules, since it applies the DataSqueezer algorithm, which generates independent rules.

- it has linear complexity. Complexity analysis of the MetaSqueezer system shows that it is liner in respect to the number of examples in the data sets, i.e., O(*s*), where *s* is the number of examples. The theoretical complexity analysis of the system is shown in section 4.2.1.
- it generates highly user friendly results. The system provides very easy to understand representation of generated meta-rules, which consists of attribute and selector tables. Such tables are inferred directly from the rules. The tables are generated using a procedure described in section 5.5.1.

The above properties of the system show its high applicability to real-life problems that concern high volumes of input data, and require providing simple and easy to comprehend results. The system is flexible, since it can be applied to all types of attributes, is noise and missing values resistant, and provides very accurate and compact rules. These properties are shown in the subsequent sections.

4.2.1 Theoretical Complexity Analysis

The complexity of the MetaSqueezer system is determined by complexity of the DataSqueezer algorithm. Assuming that *s* is the number of examples in the original data set, the MetaSqueezer system divides the data into *n* subsets, of *s*/*n* size. In the DM step, the MetaSqueezer system uses DataSqueezer algorithm *n* times, which gives total complexity of *n*O(*s*/*n*) = O(*s*), since complexity of the DataSqueezer algorithm is O(*s*). In the MM step, the DataSqueezer algorithm is run once with the data of size O(*s*). Thus, complexity of the MetaSqueezer system is O(s) + O(s) = O(s). The complexity of the preprocessing step was omitted, since the reported in the literature complexity results always omit this step in calculations. As the results of the theoretical analysis, the MetaSqueezer system has linear complexity. This result is also supported experimentally in section 4.3.1.4.

4.3 Experimental Evaluation

The MetaSqueezer system was extensively benchmarked to show its validity, simplicity, efficiency, and flexibility. The benchmarking procedure is very similar to the procedure described for the DataSqueezer algorithm. It was tested on 20 standard benchmarking data sets. The data sets were obtained from the University of California Irvine (UCI) Machine Learning Repository (Blake and Merz, 1998), the UCI KDD Archive (Hettich & Bay, 1999), and from the StatLib project data sets repository (Vlachos, 2000). Using standard benchmarking data sets enables direct comparison of performance of the MetaSqueezer system and other ML algorithms that generate similar results. The benchmarking setup was developed mainly to perform comparison between the MetaSqueezer system and the DataSqueezer algorithm; e.g., both were tested on the same data sets. The comparison between the DataSqueezer and MetaSqueezer is a vital part of the benchmarking tests, since it provides validation for the development of the system. Since the MetaSqueezer system uses the

DataSqueezer algorithm, a strong reason, proven via tests, should be provided to validate creation of the MetaSqueezer system.

The detailed description of the data sets is given in Table 7. Both, percent of missing values and percent of inconsistent examples refer to already discretized training sets. The number of subsets column refers to the number of subsets generated from the initial training set during the preprocessing step of the MetaSqueezer system. This is a user-defined number, and depends mostly on the size of the data, i.e., the larger the data the bigger the number of subsets should be used. The number of subsets for the 20 data sets was between 2 for the smallest *tea* data set, and 10 for the largest *led*, *sat*, *adult*, and *cid* datasets.

The benchmarking tests are characterized by strong diversity of training sets, including their size, both in terms of number of examples and attributes, number of classes, attribute types, and amount of missing values and noise in the data. The range of the tests is characterized by:

- the size of training data sets: between 151 and 200K examples,
- the size of testing data sets: between 15 and 565K examples,
- the number of attributes: between 5 and 61,
- the number of classes: between 2 and 10,
- the attribute types: nominal, discrete numerical, and continuous numerical,
- the percentage of examples with missing values: between 0 and 52.3,
- the percentage of inconsistent, noisy examples: between 0 and 66.3,

- the number of training data subsets: between 2 and 10.

The diversity of the tests ensures that evaluation of the MetaSqueezer system, which is performed based on comparison with results achieved by other state of the art inductive ML algorithms including the DataSqueezer algorithm, is comprehensive and strong.

4.3.1 Comparison of MetaSqueezer with Other Algorithms

The tests compare accuracy of the rules, number of rules and selectors, and execution time. The MetaSqueezer system was compared to DataSqueezer algorithm (Kurgan and Cios, 2002a; Kurgan and Cios, 2003a), CLIP4 algorithm (Cios and Kurgan, 2002a), and 33 other inductive ML algorithms, for which the results were published in (Lim et al., 2000). This study reports results on the first 16 data sets described in Table 7. The remaining 4 data sets were chosen because of their larger size, and incorporation of larger amount of missing values. The last, *cid*, data set was also used to perform experimental complexity analysis of the MetaSqueezer system.

4.3.1.1 Accuracy

To evaluate accuracy of the MetaSqueezer system, the verification test was performed. The summary of the test results is shown in Table 19. The table shows minimum and maximum accuracy for the 33 inductive ML algorithms (Lim et al., 2000), accuracy for the CLIP4 algorithm, and accuracy, sensitivity, and specificity for the MetaSqueezer system and the DataSqueezer algorithm (Kurgan and Cios, 2003a). Only the MetaSqueezer algorithm works using the MM concept, which required dividing the training set into subsets. The remaining algorithms use original data sets, which were not divided into subsets, as input.

Table 19. Accuracy results for the MetaSqueezer, DataSqueezer, CLIP4, and the other 33 ML algorithms

set	Reported accuracy (Lim et al., 2000)		CLIP4 accuracy (Cios and Kurgan, 2002a)	DataSqueezer			MetaSqueezer		
	max	min		mean accuracy	mean sensitivity	mean specificity	mean accuracy	mean sensitivity	mean specificity
bcw	97	91	95	94	92	98	93	97	85
bld	72	57	63	68	86	44	70	93	38
bos	78	69	71	70	70	88	71	70	86
cmc	57	40	47	44	40	73	47	43	72
dna	95	62	91	92	92	97	90	89	95
hea	86	66	72	79	89	66	79	87	70
led	73	18	71	68	68	97	69	69	97
pid	78	69	71	76	83	61	75	83	59
sat	90	60	80	80	78	96	74	73	95
seg	98	48	86	84	83	98	81	81	97
smo	70	55	68	68	33	67	67	33	69
thy	99	11	99	96	95	99	96	86	99
veh	85	51	56	61	61	88	60	59	87
vot	96	94	94	95	93	96	94	92	99
wav	85	52	75	77	77	89	78	78	89
tae	77	31	60	55	53	79	52	51	76
MEAN	**83.5**	**54.6**	**74.9**	**75.4**	74.6	83.5	**74.8**	74.0	82.1
set	algorithm (accuracy) (reference)			mean accuracy	mean sensitivity	mean specificity	mean accuracy	mean sensitivity	mean specificity
adult	NBTree (84) (Kohavi, 1996)			82	94	41	81	95	33
	C4.5 (84.5), C4.5-auto (85.5), Voted ID3-0.6 (84.4), T2 (83.2), 1R (80.5), CN2 (84), HOODG (83.2), FSS Naive Bayes (**86**), IDTM (85.5), Naive-Bayes (83.9), NN-1 (78.6), NN-3 (79.7), OC1 (85) (Blake & Merz, 1998)								
cid	C4.5 (95.2), C5.0 (95.3), C5.0 rules (95.3), C5.0 boosted (**95.4**), Naïve-Bayes (76.8) (Hettich & Bay, 1999)			91	94	45	90	93	49
forc	[NN-backprop (**70.0**), Linear Discriminant Analysis (58.0)] (Blackard, 1998)			55	56	90	55	36	89
mush	[C4.5 (**100**), NBTree (96.5)] (Kohavi, 1996)			100	100	100	100	99	100
	[STAGGER (95)] (Schlimmer, 1987)								
	[HILLARY (95)] (Iba, Wogulis & Langley, 1988)								
MEAN	mean best: **87.9** means worst: **77.1**			**82.0**	86.0	69.0	**81.5**	80.8	67.8

The average accuracy of the MetaSqueezer system for the first 16 data sets is 74.8%, while for the DataSqueezer algorithm is 75.4%. The difference in accuracy is insignificant, showing that the results generated by the MetaSqueezer system achieve the same accuracy as the results generated by the DataSqueezer algorithm. Also, the difference in accuracy between the two algorithms for the latter four data sets is insignificant. Both generate very accurate rules for them. This is a significant achievement of the MetaSqueezer system, since it provides more compact rules (when compared to the DataSqueezer algorithm), which is shown in the next section, without trading their accuracy.

To situate the MetaSqueezer system among other inductive ML algorithms we compare it with the 33 algorithms from the (Lim et al., 2000) study. The POLYCLASS algorithm (Kooperberg et al. 1997) achieved the highest mean accuracy of 80.5% among the 33 algorithms. Also, the (Lim et al., 2000) calculated statistical significance of error rates, which shows that a difference between the mean accuracies of two algorithms is statistically significant at the 10% level if they differ by more than 5.9 %. Analysis of the results shows that the MetaSqueezer system's accuracy is within the range of POLYCLASS, as well as all other ML algorithms, including CLIP4 and DataSqueezer. The accuracies achieved by the MetaSqueezer system place it among the best inductive ML algorithms. The mean sensitivities and specificities achieved by the MetaSqueezer system have high and comparable values. This is a very desirable property, which means that the generated rules describe correctly all classes in the data, i.e., the rules are not biased towards describing, for example, only classes that are described by majority of examples. The same property was exhibited by the DataSqueezer system. The MetaSqueezer system again preserved a positive property originally exhibited by the DataSqueezer algorithm.

The results show that the MetaSqueezer system generates very accurate rules, which means that it is characterized by high validity. The validity of the MetaSqueezer system is comparable to validity achieved by the DataSqueezer algorithm.

4.3.1.2 Simplicity and Efficiency

The simplicity and efficiency related tests are used to validate if the MetaSqueezer system generates compact and easy to understand rules, and if it generates them quickly. The tests report number of rules, number of selectors, and execution time. Table 20 shows the results for the MetaSqueezer system, the DataSqueezer algorithm, and the other ML algorithms (Kurgan and Cios, 2003a). The MetaSqueezer system is compared with results achieved the DataSqueezer algorithm, by 33 algorithm reported in the (Lim et al., 2000), and with results achieved by the CLIP4 (Cios and Kurgan, 2002a). For the 33 algorithms, the median number of rules (the authors reported the number of tree leaves for 21 decision tree algorithms) and the maximum and minimum execution time, as it was reported by the authors, is given. Additionally, for the MetaSqueezer, DataSqueezer, and CLIP4 number of selectors per rules is reported. The last measure enables direct comparison of complexity of generated rules.

Table 20. Number of rules and selectors, and running time results for the MetaSqueezer, DataSqueezer, CLIP4, and the other 33 ML algorithms

set	Reported accuracy (Lim et al., 2000)			CLIP4 (Cios and Kurgan, 2002a)				DataSqueezer				MetaSqueezer			
	mean CPU time		median # of leaves					mean time [s]	mean # rules	mean # select	# select/ rule	mean time [s]	mean # rules	mean # select	# select/ rule
	min [s]	max [h]		mean time [s]	mean # rules	mean # selectors	# select /rule								
bcw	4s	2.7	7	5.1	4.2	121.6	29.0	0.2	4.5	12.8	2.8	0.7	6.3	12.3	1.9
bld	5s	1.5	10	6.6	9.7	272.4	28.1	0.1	3.4	14.0	4.1	0.3	2.6	7.7	3.0
bos	9s	5.5	11	35.8	10.5	133.5	12.7	0.4	19.8	107	5.4	0.9	17.9	56.3	3.1
cmc	12s	23.9	15	46	8	60.7	7.6	1.3	20.2	70.5	3.5	1.6	17.4	42.1	2.4
dna	2s	475.2	13	662.8	8	90	11.3	12.5	39.0	97.0	2.5	22.8	34.0	53.0	1.6
hea	4s	3.3	6	2.2	11.6	192.3	16.6	0.1	4.7	17.1	3.6	0.3	1.9	3.7	1.9
led	1s	12.4	24	166.4	41	189	4.6	3.8	51	194	3.8	4.9	51	141	2.8
pid	7s	2.5	7	5.8	4	64.1	16.0	0.2	1.8	8.0	4.4	0.6	2.1	9.3	4.4
sat	8s	73.2	63	3696.7	61	3199	52.4	21.3	57	257	4.5	24.0	55	104	1.9
seg	28s	75.6	39	614.6	39.2	1169.9	29.8	6.1	57.3	219	3.8	6.4	50.7	89.3	1.8
smo	1s	3.8	2	90.5	18	242	13.4	1.1	6	12	2.0	1.3	3	11	3.7
thy	3s	16.1	12	164.2	4	119	29.8	1.3	7	28	4.0	1.6	6	6	1.0
veh	14s	14.1	38	45.3	21.3	380.7	17.9	1.2	23.7	80.2	3.4	1.5	22.4	41.4	1.8
vot	2s	25.2	2	4.4	9.7	51.7	5.3	0.1	1.4	1.6	1.1	0.4	1	1	1.0
wav	4s	4.3	16	43.1	9	85	9.4	0.4	22	65	2.9	0.4	16	16	1.0
tae	6s	10.2	20	0.7	9.3	273.2	29.4	0.2	21.2	57.2	2.7	0.3	14.7	27.8	1.9
MEAN	**6.9 s**	**46.8 h**	**17.8**	**5 min 49.4 s**	**16.8**	**415.3**	**19.6**	**3.1 s**	**21.3**	**77.5**	**3.4**	**4.3 s**	**18.9**	**38.9**	**2.2**
adult	N/A			16839	72	7561	105.0	399.4	61	395	6.5	231.8	19	64	3.4
cid	N/A			---	---	---	---	5,938.0	15	95	6.3	6,115.4	6	34	5.7
forc	N/A			21542	63	2438	38.7	582.0	59	2105	35.7	416.0	33	699	21.2
mush	N/A			257.4	2	18	9.0	0.1	8	21	2.6	0.4	6	16	2.7
MEAN	N/A			---	---	---	---	**1729.9**	**35.8**	**654.0**	**12.8**	**1,690.9**	**16.0**	**203.3**	**8.3**

The mean number of rules generated by the MetaSqueezer system is 18.9, while for the DataSqueezer algorithm is 21.3. The median number of tree leaves, which is equivalent to the number of rules, for the 21 tested decision tree algorithms, was reported as 17.8 (Lim et al., 2000). The number of rules generated by the CLIP4 algorithm is 16.8. For the latter four data sets, the MetaSqueezer system also achieves low number of generated rules. The number of rules generated by the MetaSqueezer system is comparable to the reported results, and most importantly, lower than the number of rules generated by the DataSqueezer algorithm. For the latter four data sets, the difference is significant, since the MetaSqueezer system generates on the average over 50% less rules. This difference was achieved for largest considered data sets, which shows that the user can potentially obtain smaller rule sets for large data sets. This result confirms one of the main advantages of the MetaSqueezer system, namely that it generates very simple and easy to understand knowledge, in terms of the number of rules.

Next, the MetaSqueezer system was analyzed in terms of the number of selectors, and the number of selectors per rule, which it generates. The number of selectors per rule generated by the system for the 16 data sets is 2.2. It means that on average each rule generated by the MetaSqueezer system involves only 2.2 attribute-values pairs in the description of the rule. This is significantly less than the number of selectors per rule achieved by the DataSqueezer algorithm (35% less), and the CLIP4 algorithm (almost 90% less). This is primarily caused by application of the MM concept where the meta-rules, which are reported here, are generated from meta-data. The remaining algorithms generate rules directly from the input data. This is yet another confirmation of the one of the main advantages of the MetaSqueezer system, namely that it generates very simple and easy to understand knowledge, in terms of achieving high compactness of the rules expressed in the small number of selectors in the rule description. Both, low number of rules, and very small number of selectors within each rule, together with very simple format of generated rules, i.e., production rules, make the results generated by the system very easy to comprehend, evaluate, and use. This is a very significant advantage of the MetaSqueezer system. It validates usefulness of the system for real-life applications concerning learning from supervised data, and assures that the results generated by the system are very simple.

MetaSqueezer's execution time for the 16 datasets was 4.3 seconds. The minimum execution time reported in (Lim et al., 2000) was 5 seconds for C4.5 algorithm (Quinlan, 1993; Quinlan 1996). The results show that the MetaSqueezer system is faster than any of the 33 ML algorithms. The only faster algorithm is the DataSqueezer algorithm which achieved execution time of 3.1 seconds. The MetaSqueezer system is build using the DataSqueezer algorithm and it is not surprising to see that both perform comparably well. The slightly worse results achieved by the MetaSqueezer system can be explained by additional computation overhead connected with dividing the data into subsets. The MetaSqueezer system generates the knowledge very efficiently. It is also interesting to note that the POLYCLASS algorithm, which achieved the best accuracy, had a mean execution time of 3.2h.

4.3.1.3 Flexibility

The tests show that the MetaSqueezer system is characterized by high validity (accuracy), simplicity and efficiency. Since these results were achieved for a diverse range of the training sets, the system is also highly flexible. It achieves excellent results for data sets that include large number of examples with missing values (e.g., *adult*, *cid*, *mush* data sets), large number of noisy examples (e.g., *bld*, *cmc*, *led*, *pid*, *smo*, *tae*, and *adult* data sets), and both large number of missing and noisy examples (e.g., *adult* data set). Also, the system is flexible since it achieves excellent performance on the test sets that incorporate all three types of attributes: nominal, discrete numerical, and continuous numerical. Thus, the MetaSqueezer system is also characterized by very high flexibility.

4.3.1.4 Experimental Complexity Analysis

The MetaSqueezer system was experimentally tested to provide additional validation of its linear complexity. As in the experimental complexity analysis performed with the DataSqueezer algorithm, the tests were performed using the *cid* data set, Table 7. The data set includes 40 attributes and 300K of data samples, which assures accuracy of the complexity analysis. The training part of the original data set was used to derive training sets for the complexity analysis. The training sets were derived by selecting a number of examples from the beginning of the original training set. A standard procedure of using training sets of doubled size for each test, to verify if the execution time was also doubled, was used. The results are summarized in Table 21 (Kurgan and Cios, 2003a). The table shows time ratios for the subsequent experiments, along with the results of the verification test, and the number of generated rules and selectors.

Table 21. Summary of experimental complexity analysis results for the MetaSqueezer system

Train data size	1000	2000	4000	8000	16000	32000	64000	128000	199523
Train data size (ratio)	---	2	2	2	2	2	2	2	1.56
Time	65msec	1sec 27msec	2sec 31msec	4sec 72msec	10sec 44msec	24sec 71msec	55sec 43msec	2min 31sec 71msec	4min 33sec 01msec
Time - Ratio	---	1.95	1.81	2.04	2.21	2.34	2.24	2.74	1.80
Accuracy	93.6	90.6	90.3	92.0	92.5	91.7	90.5	90.9	90.2
Sensitivity	99.6	94.9	93.8	96.3	97.2	95.9	93.7	94.1	93.0
Specificity	1.9	25.7	37.1	26.5	20.6	28.1	41.8	41.7	49.0
# Rules	38	11	14	17	18	13	11	11	6
# Selectors	173	67	76	106	98	66	72	67	34

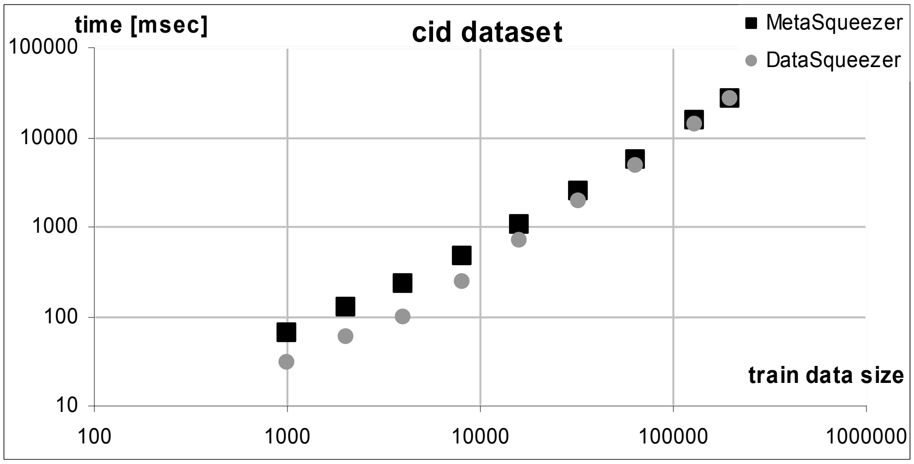


Figure 16. Relation between execution time and input data size for the MetaSqueezer and the DataSqueezer algorithms

The results are visualized in Figure 16. The figure shows a graph of relation between execution time and size of the data for both MetaSqueezer system and DataSqueezer algorithm. The results for both algorithms are shown to compare their performance. The graph uses logarithmic scale on both axes to help visualize data points of low values.

The results show linear relationship between the execution time and the size of input data, for both MetaSqueezer and DataSqueezer, and thus agree with the theoretical complexity analysis. The linearity is confirmed by observing that the time ratio is always close to the data size ratio. The difference in ratios between the time and training data size is shown in Figure 17. It shows that the ratio difference does not grow with the growing size of input data.

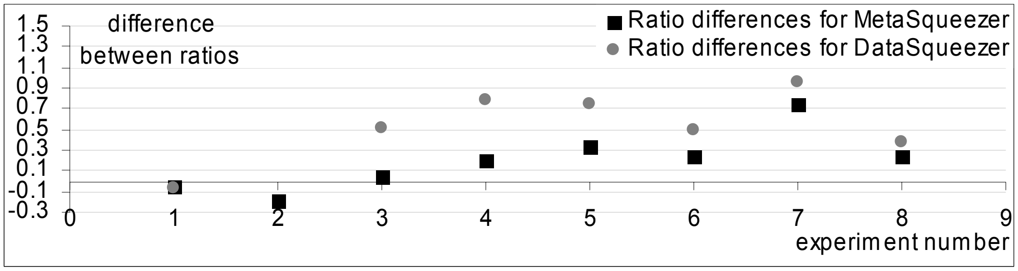


Figure 17. The difference in ratios between the time and training data size

The slightly higher values of time ratio can be explained by computational overhead for preparation of the input data. However, it is important that the overhead does not increase with the increase of the input data size. The graph in Figure 16 shows that the MetaSqueezer system has additional overhead connected with division of input data into subsets, which becomes insignificant with the growing size of the input data. This property is exhibited by initially lower execution time of the DataSqueezer algorithm, where the difference in execution time between the MetaSqueezer system and the DataSqueezer algorithm shrinks with growth of the size of data.

4.3.1.5 Summary of Experimental Evaluation

To summarize, the three main advantages of the MetaSqueezer system are the compactness of the generated rules, low computational cost, and user-friendliness of the generated results. Experimental results show that the system strongly exhibits all four qualities of supervised inductive ML algorithms, i.e., accuracy, simplicity, efficiently, and flexibility. It generates rules very efficiently since it is characterized by linear complexity. It also generates very simple rules that perform with high accuracy. Still, it is sufficiently flexible to handle all types of attributes, and training sets that incorporate high number of noisy and missing values examples. Using the results of the analysis of the system's qualities, the advantages of the MetaSqueezer system are verified. It generates both compact and user-friendly rules since it generates small number of short rules, and the rules are in easy to comprehend format, i.e., production rules. Table 11 summarizes the results of the experimental evaluation of the MetaSqueezer system.

Table 22. Summary of the benchmarking tests for the DataSqueezer algorithm

accuracy	simplicity	efficiency	flexibility
high	very high	very high	highly flexible: handles all attribute types noise resistant missing values resistant

The results place the MetaSqueezer system among the best inductive ML algorithms. The reader is encouraged to inspect Appendix C for detailed results, which include graphs and detailed reports from 10 fold cross validation runs.

Chapter 5

5 Application of MetaSqueezer System to Analysis of Cystic Fibrosis Data

In this chapter we describe application of the system to analysis of data describing patients with cystic fibrosis. The goal was to provide answers for two data mining goals specified by the owners of the data: finding factors related to the pace of the disease development, and to different type of the disease.

5.1 Introduction

The MetaSqueezer system was used to perform analysis of medical data. The project was carried out using the DMKD process model, which is described next.

Several researchers have described a series of steps that constitute the KD process. They range from very simple models, incorporating few steps that usually include data collection and understanding, data mining, and implementation, to more sophisticated models like the nine-step model proposed by Fayyad et al. (Fayyad et al., 1996c). This project applies the six-step DMKD process model as defined by (Cios et al., 2000a; Cios, 2001; Cios and Kurgan, 2002b). The advantage of this model is that it is based on an industry initiated study that led to the development of an industry- and tool-independent DM process model (Wirth and Hipp, 2000; CRISP-DM, 2001). The model's usefulness for this project is supported by its successful applications to several, mostly medical, problem domains (Sacha et al., 2000; Cios et. al, 2000a; Cios et. al., 2000b; Kurgan et al., 2001). The goal of designing the DMKD process model is to develop a set of processing steps that would be followed by practitioners when executing DMKD projects. The purpose of such design is to help plan, work through, and reduce the overall costs of the project by outlining the DMKD process, and by describing procedures performed in each of the steps. The DMKD process model describes a range from problem specification to deployment of the results, i.e. discovered knowledge.

The six-step DMKD process is described as follows:

1. Understanding the problem domain. In this step the project is defined, including definition of objectives, and learning domain specific terminology and methods. A high-level description of the problem, including the requirements and restrictions is analyzed. The project goals are translated into DMKD goals and the project plan, which includes selection of suitable DM tools, is prepared.
2. Understanding the data. This step includes collection of the data, and decision regarding which data will be used (including its format and size). Next, initial data exploration is performed to verify usefulness of the data in respect to the goals identified in step 1.

3. Preparation of the data. In this step, the data that will be used as input for DM tools in step 4 is chosen. The step may involve sampling of data, performing correlation and significance tests, data cleaning like checking of completeness of data examples, assigning classes to data examples, removing or correcting noise, missing values, etc. The cleaned data can be further processed by feature selection and extraction algorithms (to reduce dimensionality), by derivation of new attributes (say by discretization), and by summarization of data (data granularization). New data records, meeting specific input requirements of the given DM tools, are formed.
4. Data mining. This step applies DM tools to discover new information from the data prepared in step 3. First, the training and testing procedures are designed. Next, the data model is constructed using one of the chosen DM tools, and the generated data model is verified by using testing procedures. Data mining tools include many types of algorithms, e.g., machine learning, rough and fuzzy sets, Bayesian methods, evolutionary computing, neural networks, clustering, association rules etc. Detailed description of these algorithms, together with specifications of their application areas, can be found in (Cios et al., 1998).
5. Evaluation of the discovered knowledge. This step includes understanding of the results, checking whether the new information is novel and interesting, interpretation of the results, and checking their impact on the project goals. Approved models are retained.

6. Using the discovered knowledge. This step consists of planning where and how the discovered knowledge will be used.

The just described DMKD process model is visualized in Figure 18, after (Cios and Kurgan, 2002b).

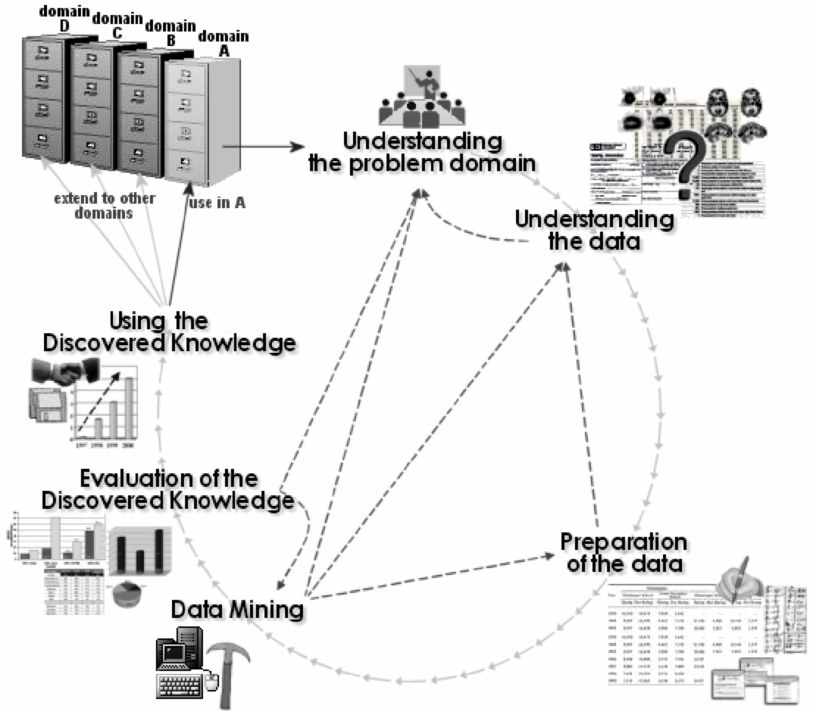


Figure 18. The six-step DMKD process model

The important issues are the iterative and interactive aspects of the process. Since any changes and decisions made in one of the steps can result in changes in later steps, feedback loops may be necessary. The feedback paths are shown by dashed lines in Figure 18. They are by no means exhaustive.

Following, the discussion concerning significance of the project is provided. Next, the project is described using the six step DMKD process model.

5.1.1 Significance

The project concerns analysis of cystic fibrosis data. The data set is temporal in nature, and as such needs specific learning tools. The MetaSqueezer system can be easily used for analysis of temporal data, and thus was chosen to perform analysis.

Despite extensive literature search only one other application of inductive ML techniques to analysis of temporal medical data was found. A system by (Karimi and Hamilton, 2000) discovers very limited temporal relations using Bayesian Networks and inductive machine learning algorithm C4.5 (Quinlan, 1993). The two algorithms are used to find relations between temporarily consecutive records but without generalizing temporal rules for the entire period of time. Also, the MM concept was not used in the system.

Although the MetaSqueezer system does not discover any temporal relationships, it can be used to derive non temporal patters, in terms of production rules, that describe the data across the time, but using meta-data that describes data within particular temporal intervals. Also, other important factors, like efficiency of the system and compactness of results that it generates, decided on application of the system in this project.

Medical applications often aim to describe patterns in disease development and to predict therapy effectiveness. The application of the MetaSqueezer system can be included within the first category. Our goal is to discover patterns (important factors) that are associated with different paces of development of cystic fibrosis disease. As a secondary goal, the system was used to find important factors associated with particular gene types that cause cystic fibrosis.

5.2 Understanding the Problem Domain

Since the MetaSqueezer system is applied to medical data describing cystic fibrosis (CF) patients, first the disease is introduced. CF is a genetic disease affecting approximately 30,000 children and adults in the United States (Cystic Fibrosis Foundation, 2002). One in 31 Americans, and one in 28 Caucasians, which translates into more than 10 million people carry the defective gene causing CF. They do not exhibit the symptoms, and thus they do not know about the disease. An individual must inherit a defective copy of the CF gene from each of the parents to become affected. Statistically, when two carriers conceive a child, there is a 25 percent chance that the child will have CF, a 50 percent chance that the child will be a carrier, and a 25 percent chance that the child will be a non-carrier.

CF is a deadly disease that affects multiple systems, including the respiratory system, digestive system, endocrine system, and reproductive system. It has multiple symptoms including very salty-tasting skin, persistent coughing, wheezing or pneumonia, excessive appetite but poor weight gain, and bulky stools. CF is diagnosed usually by the sweat test, which measures amount of salt in the sweat. A high chloride level indicates that a person has CF.

The treatment of CF depends upon multiple factors like stage of the disease and which organs are involved. In case of the most severely affected organ, the lungs, the disease is treated usually by chest physical therapy, and antibiotics, which are used to treat lung infections. When CF affects the digestive system, the patients are required to eat an enriched diet and take replacement vitamins and enzymes (Cystic Fibrosis Foundation, 2002).

One of fundamental assumptions in the project was for the medical staff to provide only the necessary minimum background knowledge to us. The main reason for that was to assure that the research will not be biased toward finding solutions that would confirm accuracy of the MetaSqueezer system based on the domain knowledge. By following this assumption, a true evaluation of the system was provided.

The project goal is to perform analysis of data concerning patients with CF. Although CF is an extensively studied disease, the amount of available data and support of well qualified medical staff were important and sufficient factors that lead to starting the project. The predicted outcome of the investigation was to provide new findings that may advance knowledge about the disease and its treatment methods. Also, the investigation was an opportunity to confirm correctness of the system, for example by finding results, which are already described in literature.

The two goals defined by clinicians were:

- task 1 is to discover patterns (important factors) that influence the pace of the development of CF. Although CF affect multiple systems, the one system that is used to indicate the progress of the disease is the pulmonary system. It is interesting that for some patients the disease progresses very fast, while for others it progresses relatively slow. There are some known factors that are related to the pace of the disease, but still much of them probably remain unknown. Thus, the first goal was to discover such factors that are related to different, predefined paces of the disease development based on historical data concerning CF patients.
- task 2 is to discover important factors that are related to particular kinds of CF. CF is a genetic disease. As for such, genotypes related to the disease are described. Our task was to find factors that are related to different, predefined types of CF based on historical data concerning CF patients.

The CF data is temporal in nature. It describes several hundreds of CF patients in time. For each patient multiple visits are recorded. Most of the patients are monitored between their birth and death. For each visit multiple attributes describing demographical information, various lab results, and diagnostic information are recorded. It is a known fact that the data describes different relationships depending on the stage of the disease. Thus, any investigation that uses such data must be able to separate the data into subsets corresponding to particular stages of the disease.

In the next step the two goals were redefined into mining goals:

- Task 1. Such mining goal can be defined as a supervised inductive learning task. Categorical class attribute must be defined, which will group the data into subsets corresponding to patients who exhibit different pace of the disease development. Also, since the data is temporal, another attribute will be used to divide the data across the time domain. The attribute will describe the stage of the disease based on the status of a pulmonary function test.
- Task 2. Such goal can be also defined as a supervised inductive learning task. Thus, a categorical class attribute that describes patients in terms of the particular type of CF they have must be defined. Next, similarly as for the first task, the data must be divided in temporal manner by using attribute(s) describing patient's lung functions.

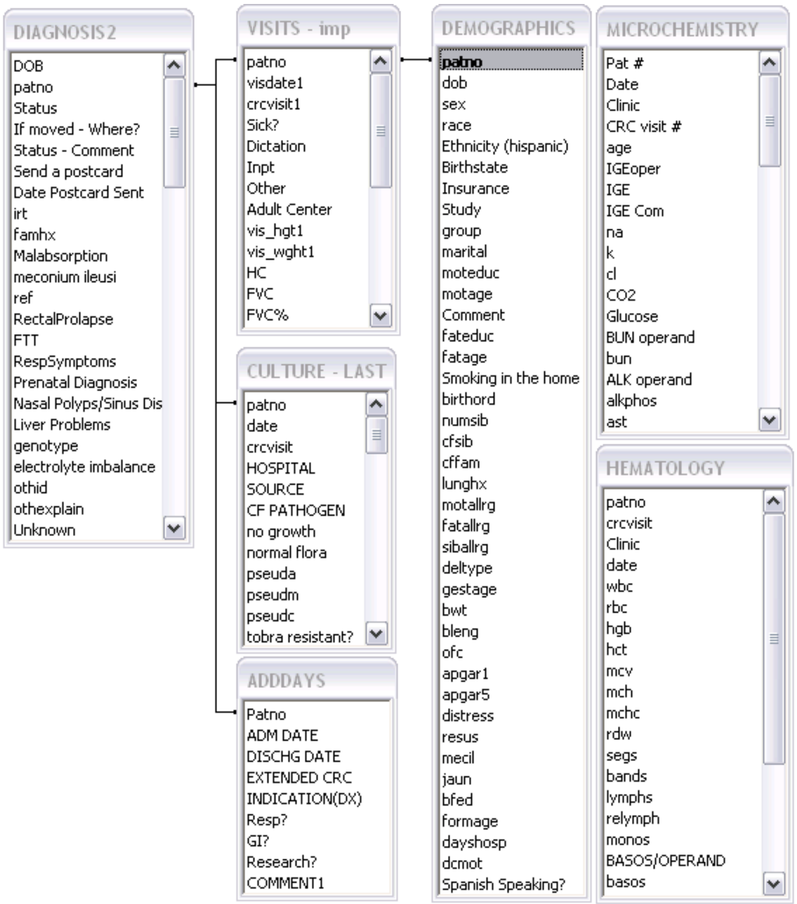
After analyzing both tasks, the MetaSqueezer system was identified as a DM tool capable to provide desired results. There are four factors that decided about choosing the system:

- it generates very simple to understand rules. This project requires physicians to be able to analyze and comprehend the rules. This is necessary to evaluate and use the results. The system not only generates small number of rules, but also very compact rules, which tremendously helps in their analysis

- it is scalable. The system has linear complexity and thus can be used with large quantities of data. The estimated size of the CF data set was about 20K examples by around 200 attributes, and thus an efficient system is necessary to generate the results.
- it can handle large quantities of missing values. Since the CF data is a real-life, clinical data set, it is expected that it will contain large quantities of missing information. The MetaSqueezer system is able to handle data sets which contain significant amount of missing values since such data is always used an an input to the MM step. The DataSqueezer which is a core inductive ML algorithm used within the system is proven to generate accurate results even in presents of significant amount of missing information.
- it can handle temporal data. Although the system does not provide temporal-like knowledge representation, it can be succesfully applied to temporal data. The DM step of the MetaSqueezer system accepts multiple training data sub sets, which can represent temporally organized subsets of the original data.

5.3 Understanding the Data

The data used in this project was donated by Dr. Frank Accurso, pediatric pulmonologist from Denver Children's Hospital. It was collected starting in 1982. It includes demographical information about patients, clinical information including a variety of lab tests, and diagnoses. The data includes information about 856 patients. It was stored in the MS Access 97 using seven relational tables. The tables, together with the relational dependencies (shown as links) are shown in Figure 19.



DIAGNOSIS2
DOB
patno
Status
If moved - Where?
Status - Comment
Send a postcard
Date Postcard Sent
irt
famhx
Malabsorption
meconium ileusi
ref
RectalProlapse
FTT
RespSymptoms
Prenatal Diagnosis
Nasal Polyps/Sinus Dis
Liver Problems
genotype
electrolyte imbalance
othid
othexplain
Unknown

VISITS - imp
patno
visdate1
crcvisit1
Sick?
Dictation
Inpt
Other
Adult Center
vis_hgt1
vis_wght1
HC
FVC
FVC%

CULTURE - LAST
patno
date
crcvisit
HOSPITAL
SOURCE
CF PATHOGEN
no growth
normal flora
pseuda
pseudm
pseudc
tobra resistant?

ADDDAYS
Patno
ADM DATE
DISCHG DATE
EXTENDED CRC
INDICATION(DX)
Resp?
GI?
Research?
COMMENT1

DEMOGRAPHICS
patno
dob
sex
race
Ethnicity (hispanic)
Birthstate
Insurance
Study
group
marital
moteduc
motage
Comment
fateduc
fatage
Smoking in the home
birthord
numsib
cfsib
cffam
lunghx
motallrg
fatallrg
siballrg
deltype
gestage
bwt
bleng
ofc
apgar1
apgar5
distress
resus
mecil
jaun
bfed
formage
dayshosp
dcmot
Spanish Speaking?

MICROCHEMISTRY
Pat #
Date
Clinic
CRC visit #
age
IGEoper
IGE
IGE Com
na
k
cl
CO2
Glucose
BUN operand
bun
ALK operand
alkphos
ast

HEMATOLOGY
patno
crcvisit
Clinic
date
wbc
rbc
hgb
hct
mcv
mch
mchc
rdw
segs
bands
lymphs
relymph
monos
BASOS/OPERAND
basos

Figure 19. The structure of the CF data

Detailed description of each of the tables is provided below:

- VISITS (vis) table
  - The table holds the most important information relevant to patient's visits to the clinic. One of the attributes, which is stored in the visits

table is the FEV% attribute that will be used as the time-defining attribute, as well as the class attribute.

  - Statistical information:
    - 15,199 examples
    - 26 attributes, numerical and binary
    - 183,845 missing values; 46.5% of the total number of values
    - no keys are defined
- ADDDAYS (add) table
  - The table holds administrative information, like patient admission information, kind of sickness that patient had when admitted, etc.
  - Statistical information:
    - 2,141 examples
    - 9 attributes, binary, textual, numerical, and date/time
    - 3,011 missing values; 15.6% of the total number of values
    - no keys are defined
- DEMOGRAPHICS (dem) table
  - The table holds demographical information about patients, like gender, race, ethnicity, family situation, etc.
  - Statistical information:
    - 856 examples
    - 40 attributes, binary, textual, numerical, and date/time
    - 18,317 missing values; 53.5% of the total number of values

    - "patno" is defined as the primary key

- DIAGNOSIS2 (dia) table
  - The table holds diagnostic information. Although each patient is diagnosed with CF, the tests that lead to the diagnosis are indicated, and values for theses tests are provided. The table holds two attributes, Genotype1 and Genotyp2, that will be used to define the class attribute
  - Statistical information:
    - 856 examples
    - 89 attributes, binary, textual, numerical, and date/time
    - 42,336 missing values; 55.6% of the total number of values
    - no keys are defined
- CULTURE_LAST (cul) table
  - The table holds a variety of laboratory test results
  - Statistical information:
    - 6,904 examples
    - 65 attributes, numerical and textual
    - 235,054 missing values; 52.4% of the total number of values
    - no keys are defined
- MICROCHEMISTRY (mic) table
  - The table holds a variety of laboratory test results
  - Statistical information:
    - 3,138 examples

        - 65 attributes, numerical
        - 146,307 missing values; 71.7% of the total number of values
        - no keys are defined

- HEMATOLOGY (hem) table
    - The table holds a variety of laboratory test results
    - Statistical information:
        - 2,348 examples
        - 27 attributes, numerical and textual
        - 22,043 missing values; 34.8% of the total number of values
        - no keys are defined

Because of the confidentiality issues, all identification information was removed from the data before it was used in the project. The data holds only relevant clinical information.

There are several issues with the CF data. First, as expected, it contains significant amount of missing information. For example, MICROCHEMISTRY, CULTURE_LAST, and DIAGFNOSIS2 tables contain more than half of missing information. Second, it can be also expected that since the data was inserted manually by the physicians, it will contain also substantial amount of incorrect records. Next, the tables contain different attributes: numerical, textual, and binary, which need to be handled by the learning algorithm. The tables also possibly include large quantities of irrelevant information, in terms of attributes, which may be removed before the learning process is executed. All these issues need to be addressed in the next step.

Below, the FEV%, Genotype 1, and Genotype 2 attributes are described. These attributes will be used to derive classes for both tasks, and to divide the data into temporal intervals:

- FEV% (Forced Expiratory Volume in One Second % Predicted) attribute is stored in the VISITS table. It describes the amount of air that can be forced out in one second after taking a deep breath. Since one of the most significant symptoms of CF is obstruction of lungs, the lungs function tests are very good indicator of the stage of the disease. In case of the CF data, the test result indicates the stage of the disease better than the timestamp information, because different patients are diagnosed and start treatment at different age. Another advantage of FEV% is its independence of patient characteristics like weight, age, race, etc., since these characteristics are used to compute its values. The FEV% is used as the attribute to define temporal intervals for both tasks. It is also used to define the class attribute for the first task. Since goal of this task is to discover important factors that influence the pace of CF development, several categories of the disease development pace were defined using the FEV% attribute. The choice of the FEV% attribute was suggested by Dr. Accurso who is the owner of the data set.

- Genotype 1 and Genotype 2 attributes are stored in the DIAGNOSIS2 table. The CF is caused by at least 1000 different genetic mutations, but approximately 70% of the mutations are found to be delta F508 gene, making it the most common CF mutation (Cystic Fibrosis Genetic Analysis Consortium, 1990; Cystic Fibrosis Mutation Database, 2003). The CF data includes two attributes describing the genetic mutations: Genotype 1 and Genotype 2. These attributes are used to distinguish different kinds of CF. Since the second mining task is to find important factors that are related to particular kinds of CF, combination of the two attributes will be used to provide class labels for the task. According to Dr. Accurso, four kinds of CF need to be defined: 1) both Genotype 1 and Genotype 2 are F508, 2) Genotype 1 is F508 and Genotype 2 is any other genotype, 3) Genotype 2 is F508 and Genotype 1 is any other genotype, 4) both Genotype 1 and Genotype 2 are not F508.

In summary, the data was identified to hold all information necessary to carry out the project. More specifically, several attributes that may be used to derive classes and temporal subsets for both learning tasks were identified. Also, the amount and variety of information included in the CF data gives a strong reason to believe that interesting patters may be discovered.

5.4 Preparation of the Data

Before the CF data will be used to generate rules using the MetaSqueezer system, it needs to be preprocessed. This usually involves removing or correcting noise and missing values, sampling and reorganizing the data, assigning classes to examples, identifying temporal intervals, etc. The cleaned data is later discretized since the MetaSqueezer system works only with discrete data. As with most of the DM projects, it is expected that this step consumes most of the allocated time (Cabena et. al, 1998, Cios and Kurgan, 2002b). It is well recognized that data preparation is a very important task that greatly affects the outcome of the entire project, and thus it often takes a significant portion of the total project effort

The CF data is stored in seven relational tables. As a first step, a manual data checking and cleaning was performed. It is expected that this task will consume significant amount of time since the CF data contains significant amount of errors and inconsistencies. Since the data concerns medical domain, problems are expected connected with physician's interpretation that is written in an unstructured free-text English (text fields in the database), which is very difficult to standardize and thus difficult to mine (Cios and Moore, 2002). To point out some of the problems, the values of the FEV% attribute from VISITS table should be in the range [0; 200], but some records have bigger values. Since the information stored in this attribute is critical for the MetaSqueezer system, this problem needs to be carefully corrected. Another problem is presence of null attributes. Such attributes have null values for all tuples (rows) and thus should be deleted. Almost all tables in the CF data contain attributes that are null.

Table 23. Summary of data cleaning performed with CF data

Table	Fixed inconsistencies	Fixed errors
DEMOGRAPHICS	sex : F → f, Female → f, M → m, Male → m group: nbs → NBS marital: u → Unknown, m → Married, s → Single, d → Divorced mecil: No → no, NO → no, Yes → yes, y → yes, Treated Surg → TreatedSurgically, tr surgically → TreatedSurgically, trsurg → TreatedSurgically, treated surgically → TreatedSurgically, surg → TreatedSurgically, surgery → TreatedSurgically, Treated Surgically → TreatedSurgically, Treated Med → med jaun: Unknown → unk, Untreated → unt, untreated → unt, treated → trt, No → no, NO → no, Treated → trt bfed: Yes → yes, No → no, NO → no, y → yes, YES → yes, YYES → yes, n → no dcmot: y → yes, n → no, Yes → yes, YES → yes, Y → yes, NO → no, No → no deltype: Vaginal → vagnl	**marital**: o, 1
ADDAYS	EXTENDEDCRC : n → N	NONE
CULTURE_LAST	SOURCE: SPUTUM → Sputum SOURCE2: Throat cx → Throat Cx VIRUSDETECTED: N → n, Y → y, ND → n Adeno: N → n, Y → y CMV: N → n, Y → y FLU: N → n ParafluI: N → n, Y → y ParafluII: N → n ParafluIII: N → n Rhino: N → n Coxsa: N → n Polio: N → n RSVCX: N → n RSVRAPID: N → n	VIRUSDETECTED: +, =
DIAGNOSIS2	Status : unknown → Unknown site1: other → OTHER, Other → OTHER, Oher → OTHER val2: M → m site2: other → OTHER, Other → OTHER site3: other → OTHER, Other → OTHER Geno: removed entire attribute Date of NPD: removed entire attribute Date Post card Sent: removed entire attribute	age@diagnosis: -73.9,-93.3 age@sweate3: -73.9,-94.2,-93.3 Geno: entirely NULL attribute Date of NPD: entirely NULL attribute Date Post card Sent: only a single date 10/19/2001 Genotypes2: comment sweatna1: 878
HEMATOLOGY	pivka: NEG → Neg	pivka: 77,0.4,0.99 IIAg: NEG empty records for PatNo: 1948, 514, 97, 112, 893, 194, 665, 1940, 248
MICROCHEMISTRY	BUNoperand: S → s Ca:mg/dL;MEQ/L: meq/l → MEQ/L VitAmeas: UG/DL → ug/dl, MCG/DL → mcg/dl, ug/dL → ug/dl IGECom: removed entire attribute	empty record for PatNo: 863, 642, 561, 956, 640 IGECom: entirely NULL attribute
VISITS	NONE	FEV%: values above 150: 151, 158, 164, 166, 808, 982 Inpt: delete examples with Inpt=1 PulseOximetry: delete values > 100

The two main manual operations were performed to clean the CF data. First, the consistency of attributes was corrected by merging together their corresponding values. Such operation is caused by inconsistent format of inserted data, and must be performed since inductive ML algorithms do not recognize such correspondence. Next, erroneous values of attributes were identified and removed. This was performed for all of the tables, and is summarized in Table 23.

After the data was cleaned, it was converted into a single relational table. In order to merge the seven tables several join operation were performed. The tables were merged in pairs, where results of one join operation were merged with next table. The following procedure was applied to generate a single table out of the seven original tables:

1. join on DEMOGRAPHICS and DIAGNOSIS2 with "patno (dem) EQUAL TO patno (dia)"
2. join on VISITS-imp and the previous join result with with "patno (vis) EQUAL TO patno (dem)"
3. join on the previous join result and ADDDAYS with with "patno (vis) EQUAL TO Patno (add) AND visdate1 (vis) CLOSEST TO ADMDATE (add)"
    - additional conditions were: "visdate1 (vis) ≤ ADMDATE (add)" AND "ADMDATE (add) – visdate1 (vis) ≤ 90 days"
4. join on the previous line result and CULTURE-LAST with with "patno (vis) EQUAL TO patno (cul) AND visdate1 (vis) CLOSEST TO date (cul)"

- additional conditions: "|date (cul) – visdate1 (vis)| ≤ 90 days"

5. join on the previous line result and MICROCHEMISTRY with with "patno (vis) EQUAL TO Pat. (mic) AND visdate1 (vis) CLOSEST TO Date (mic)"
   - additional conditions: "|Date (mic) – visdate1 (vis)| ≤ 90 days"
6. join on the previous line result and HEMATOLOGY with with "patno (vis) EQUAL TO patno (hem) AND visdate1 (vis) CLOSEST TO date (hem)"
   - additional conditions: "|date (hem) – visdate1 (vis)| ≤ 90 days"
7. join on the previous line result and PERCENTILES with with "patno (vis) EQUAL TO patno (per) AND visdate1 (vis) EQUAL TO date (per)"
   - this join operation concern a new data table that is described later in the section.

The next data preparation step consists of removing attributes that are irrelevant to the performed tasks. After consultation with Dr. Accurso, the attributes listed in Table 24 were removed. The attributes were found either irrelevant from the medical point of view or containing too much noise or missing information to be used for learning.

After removal of attributes, temporal intervals and class labels are generated. Finally, the data is discretized using both the manual discretization, and F-CAIM algorithm (Kurgan and Cios, 2003b). The outcome of this step are two relational tables that include discrete attributes, an attribute that defines class labels, one per each table, and another attribute that defines temporal intervals used to divide data in the DM step of the MetaSqueezer system. One of the generated tables is used for the first tasks, while the other for the second task.

Table 24. List of irrelevant attributes from CF data

table name	attribute name
VISITS	visdate1, Dictation, Other, AdultCenter, FVC, FVC%, FEV1, FEF25-75, , FEF25-75% , ACT, MAC , TSF, SSF, HeightforAgeZ, WeightforAgeZ, WeightforHeightZ, HC, %IBW
DEMOGRAPHICS	Insurance, Comment, Study, Smokinginthehome, cffam, lunghx, motallrg, fatallrg, siballrg, , bleng , ofc, distress, resus, jaun, formage
DIAGNOSIS2	DOB, Ifmoved-where?, Status-Comment, othexplain, DateDiagnosis, irtdate1, irtdate2, irtdate3, swedate1, swedate2, swedate3, swedate4, Comment, GenotypeDate, Accession., GenoComment, Sex, DateofEnzymeStart, Sendapostcard, birthstate, Status, Unknown, swetamt1, val1, site1, age@sweat1, swetamt2, val2, site2, age@sweat2, swetamt3, val3, site3, age@sweate3, swetamt4, val4, site4, age@sweat4, SweatTestOriginal, GenotypeOriginal, GenotypeSite, GENOGROUP, CauseofDeath, dateofEnzymestartoperand, TomNemeth?, NasalPotentialDifference, age@diagnosis
ADDAYS	ADMDATE, DISCHGDATE, INDICATION(DX), COMMENT1, EXTENDEDCRC
MICROCHEMISTRY	Date, Clinic, CRCvisit, age, IGEoper, BUNoperand, ALKoperand, Camg/dL;MEQ/L, VitD25-OH/Operand, VitAmeas, Gamma-Eoperand, TRYPoperand, CG/OPERAND, Lipase, TotalE, OGTT-FastingGlucose, OGTT-30minuteGlucose, OGTT-60minutesGlucose, OGTT-90minutesGlucose, OGTT-120minutesGlucose, ZPP, Gluthathione, VitAoperand, po4, ca19
HEMATOLOGY	date, crcvisit, Clinic, BASOS/OPERAND, EOS/OPERAND, CRP/Operand, pivka, IIAg
CULTURE_LAST	date, BASE, BASE2, BASE3, BASE4, BASE5, power, power2, power3, power4, power5, Comment, Comment2, crcvisit, HOSPITAL, CFPATHOGEN, PolymorphicCellsDe, EpithelialCellsDet, GramNegativeRodDe, GramPositiveRod, GramNegativeDiploc, GramNegativePleorm, GramPositiveCocci, GramPositiveCocci2, PseudomonasAerugino, PseudomonasAerugin2, HFlu, HOSPITAL2, SOURCE2, VIRUSDETECTED, Adeno, CMV, FLU, ParafluI, ParafluII, ParafluIII, Rhino, Coxsa, Polio, RSVCX, RSVRAPID, RSVTECHNIQUE
other	patno, FEV% (vis), Genotypes1 (dia), Genotypes2 (dia) because of correlation with class and time-defining attributes

The step of data preparation was highly iterative. Following, the final outcome, derived after several iterations, is described. Also, a brief description of all iterations is summarized at the end of this section.

The MetaSqueezer system can be applied to generate rules from supervised temporal data, such as CF data. The training set needs to be organized in a relational table where each column holds a single attribute, and each tuple describes a single example. In case of analysis of CF data, there are two attributes that must be included in the table:

- one that stores class labels. Class labels are usually derived from one of the data attributes and define the target concept. In case of Task 1, they describe the pace of the disease development, and in case of the task 2 they describe different types of CF. The "CF pace (cf)" attribute was generated to describe classes for task 1, while ""CF type (cf)" attribute was generated to describe classes for task 2.
- one that stores time-defining information. The MetaSqueezer system works with training set that is divided into subsets. These subsets are defined for example as temporal intervals, and as such will include data that is associated with particular stages of the CF. The same time-defining attribute, called "TemporalIntervals (cf)", is used for both tasks.

Following, the description of the class and time-defining attributes is provided.

5.4.1 The Class Attributes

These are two class attributes, one per each task. The first attribute describes pace of the disease development, which is used to define classes for task

1. The attribute was derived based on “visdate1” and “FEV%” attributes using the following procedure:

- remove all examples with missing “visdate1” or ”patno” or “FEV%”
    - “visdate1” describes the visit data, while “patno” describes unique patient number
- remove all examples for
    - “visitdate1” at patient’s age ≤ 7 years
    - all patients with 5 or less visits, because of possibility of inaccuracies for the linear curve fitting
- recalculate “visdate1” into the “dayno” that describes number of days since the first visit for each patient
- perform linear curve fitting (y=a*x+b) to determine pace of the disease development for each of the patients
    - use “dayno” as x, and “FEV%” as y coordinate
    - use slope of the linear interpolation of the above relation to determine the pace of CF.

The “CF pace (cf)” attributes is defined in Table 25.

Table 25. Definition of the "CF pace (cf)" attribute

CF pace (cf)	slope a	description	# patients	# data points
FastDegrad	[-inf, -0.01)	fast pace of degradation	68	2221
SlowDegrad	[-0.01, -0.005)	intermediate pace of degradation	64	2241
NoChange	[-0.005, 0.005)	no change	165	5847
Improv	[0005, inf)	slow pace of improvement	60	1598
Unknown	unknown	missing (no slope for a patient)	499	4613

The total number of negative slopes, which define degradation, was 227, positive slopes, which define improvement, was 130, while the total number of patients for whom the slopes were computed was 357. The first four categories from Table 25 define the examples that will be used to generate rules, while the last category defines examples that will be removed. This means that the first task is the 4-class supervised inductive learning task. The definition of the intervals of the slope was decided by Dr. Accurso.

The second class attribute describes different types of CF and is used to define classes for task 2. The attribute was derived based on “Genotype 1” and “Genotype 2” attributes, as shown in Table 26.

Table 26. Definition of the "CF type (cf)" attribute

CF type	Genotypes1	Genotypes2	# examples
Type1	F508	F508	8386
Type2	F508	not F508	5329
Type3	not F508	F508	0
Type4	not F508	not F508	969
Type5	either or both unknown		1836

The first four categories define the examples that will be used to generate rules, while the last category defines examples that will be removed. The Type2 and Type3 classes were merged together, and thus the second task is the 3-class supervised inductive learning task. The definition of the types was decided by Dr. Accurso.

5.4.2 The Time-Defining Attribute

The time-defining attribute, which is used to divide the data into subsets for the DM step of the MetaSqueezer system is derived from the FEV% attribute. There are 6 discrete values of the attribute, shown in Table 27, defined.

Table 27. Definition of the "TemporalIntervals (cf)" attribute

TemporalIntervals (cf)	> min FEV%	≤ max FEV%	# examples
1	0	40	824
2	40	60	1474
3	60	80	2281
4	80	100	2973
5	100	1000	1686
0	FEV% empty, error, <0, >1000	7282	

The first five categories define the examples that will be used to generate rules, while the last category defines examples that will be removed. This means that the training sets for both tasks are divided into 5 subsets during the DM step of the MetaSqueezer system. The definition of the intervals of the FEV% attribute was decided by Dr. Accurso.

5.4.3 Discretization

After deriving class and time-defining attributes, the data was discretized. First, each attribute was evaluated to belong to one of three categories: discrete, continuous for manual discretization, and continuous for automated discretization. The discrete attributes were left unchanged. The continuous for manual discretization attributes were discretized by Dr. Accurso. This was performed to generate discrete attribute, where their values have medical representation. The summary of manual discretization is provided in Table 28.

Table 28. The manual discretization of the CF data

attribute name	# of discrete intervals	discretization schema
dob (dia)	3	(-inf, 12/31/1981) → before82, (01/01/1982, 12/31/1992) → 82till92, (01/01/1993, present) → after92
DateOfDeath (dia)	2	any value → true, empty → false
crcvisit1 (vis)	2	0 → 0, non 0 → non0
Birthstate (dem)	2	CO → CO, non CO → notCO
moteduc (dem)	2	<=12 → 12andbelow, >12 → above12
fateduc (dem)	2	<=12 → 12andbelow, >12 → above12
gestage (dem)	2	<=38 → 38andbelow, >38 → above38
bwt (dem)	3	<2.5 → below2.5, <2.5, 3.2> → between2.5and3.2, >3.2 → above3.2
apgar1 (dem)	3	<5 → below5, <5, 8) → 5till8, >=8 → 8andabove
apgar5 (dem)	3	<5 → below5, <5, 8) → 5till8, >=8 → 8andabove
dayshop (dem)	3	<5 → below5, <5, 15> → between5and15, >15 → above15
ALB@DXINTERVIEW (dia)	3	<=2.5 → 2.5andbelow, (2.5, 3> → 2.5till3, >3 → above3
pseuda (cul)	2	0 → 0, non 0 → non0
pseudm (cul)	2	0 → 0, non 0 → non0
pseudc (cul)	2	0 → 0, non 0 → non0
Xanthamonas(psmaltophilia) (cul)	2	0 → 0, non 0 → non0
otherpseud (cul)	2	0 → 0, non 0 → non0
stapha (cul)	2	0 → 0, non 0 → non0
hflu (cul)	2	0 → 0, non 0 → non0
Aspegillus (cul)	2	0 → 0, non 0 → non0
NonTBMycobacterium (cul)	2	0 → 0, non 0 → non0
EColi (cul)	2	0 → 0, non 0 → non0
Klebsiella (cul)	2	0 → 0, non 0 → non0
Alcaligienesxylosoxidans (Achromobacter)	2	0 → 0, non 0 → non0
BranhCat (cul)	2	0 → 0, non 0 → non0
IGE (mic)	3	<100 → below100, <100,1000> → 100till1000, >1000 → above1000
alb (mic)	3	<=2.5 → 2.5andbelow, (2.5, 3> → 2.5till3, >3 → above3
age@diagnosis (dia)	4	<=0.25 → below025, (0.25,2> → 025till2, (2,7> → 2till7, >7 → above7

The continuous for automated discretization attributes were discretized using the supervised discretization algorithm F-CAIM (Kurgan and Cios, 2003b). The F-CAIM algorithm is a modification of the original CAIM algorithm, which exhibits the same properties as CAIM, but is faster. Since it is a supervised discretization algorithm, it discretizes an attribute using class labels. Thus, the algorithm discretizes each attribute twice, separately for each of the tasks. The list of attributes discretized using the F-CAIM algorithm is shown in Table 29.

Table 29. The discretization of the CF data using F-CAIM algorithm

table name	attribute name
VISITS	vis_hgt1, vis_wght1, PulseOximetry
DEMOGRAPHICS	motage, fatage
DIAGNOSIS2	HCT@DXINTERVIEW, irt1, age@irt1, irt2, age@irt2, irt3, age@irt3, swetna1, swetk1, swetcl1, swetna2, swetk2, swetcl2, swetna3, swetk3, swetcl3, swetna4, swetk4, swetcl4
MICROCHEMISTRY	na, k, cl, CO2, Glucose, bun, alkphos, ast, prot, ca, PHOS-S, VitD25-OH, prealb, rbp, vita, alphae, Gamma-E, TRYP2, lipids, cg, Zinc, GLYCOHGB, GGTP-S, IU/LGPT/ALT, LDH-S, CHOLESTEROL-S, CREAT-S, BILI-TOTAL, E/L, BetaCarotene, HemA1C, BileAcid
HEMATOLOGY	wbc, rbc, hgb, hct, mcv, mch, mchc, rdw, segs, bands, lymphs, relymph, monos, basos, eos, esr, PMNElastase
CULTURE_LAST	BASEpower, BASEpower2, BASEpower3, BASEpower4
PERCENTILES	WAZ, HAZ, WHZ

5.4.4 The Training Set for Task 1

The task 1 goal it to discover factors related to different paces of the development of CF. Thus, the training set uses “CF pace (cf)” attribute as the class attribute, and “TemporalInterval (cf)” as the attribute to divide the data into subsets during the DM step of the MetaSqueezer system. The training set was derived from the original data first by removing noise and inconsistencies, merging the seven tables, removing irrelevant attributes, defining class and time-defining attributes, and finally discretizing the remaining continuous attributes. Two more steps were performed to generate final version of the training set:

- examples that include incomplete critical information, in terms of class or temporal information, were removed. Thus, all examples that have "CF pace (cf)" = Unknown OR "TemporalInterval (cf)" = 0 were removed. Total of 11,872 examples were removed.
- examples that incorporate too many missing values were removed. This was performed to remove all examples that contain large number of missing values introduced by performing join operations. During the join, if a tuple from one table was not matched with a tuple from another table, it was padded with missing values. If an example was not matched during several subsequent joins, as a result it contains many missing values, and can be treated as an outlier and thus removed. To decide which examples will be removed, a graph that shows relationship between the number of missing values and number of examples that have the number of missing values was used, see Figure 20.

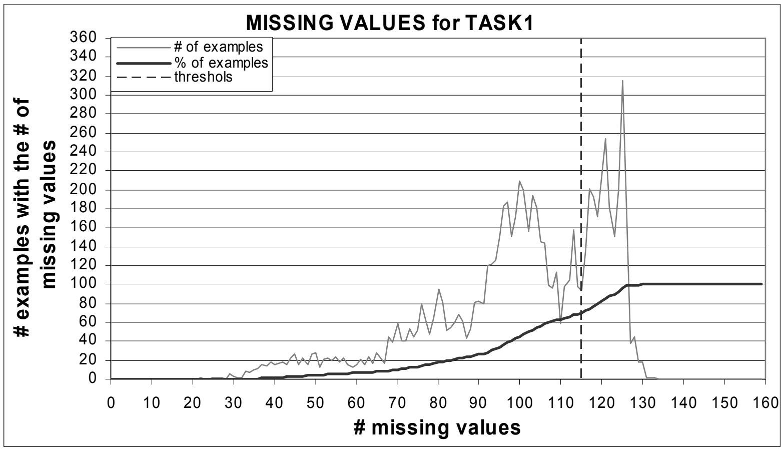


Figure 20. The relationship showing number of examples with particular number of missing values for task 1

The graph shows a peak of number of missing values above 115 missing values per example. This high peak is associated with the examples that were padded with zeros during join operations, and thus might be removed without impairing the quality of learning. The examples with more than 115 missing values were removed. Total of 2,316 examples were removed.

After applying the above preprocessing steps, the training set for task 1 consists of 5,448 examples described by 160 attributes, and with total number of values of 871,680. The number of missing values is 492,961, which constitutes 56.6 % of the entire set.

5.4.5 The Training Set for Task 2

The task 2 goal it to discover factors related to different types of CF. Thus, the training set uses "CF type (cf)" attribute as the class attribute, and "TemporalInterval (cf)" as the attribute to divide the data into subsets during the

DM step of the MetaSqueezer system. The training set for task 2 was derived the same way as the training set for the task 1, except the last two steps for which the description follows:

- examples that include incomplete critical information, in terms of class or temporal information, were removed. Thus, all examples that have "CF type (cf)" = Type5 OR "TemporalInterval (cf)" = 0 were removed. Total of 7,538 examples were removed.
- examples that incorporate too many missing values were removed. Similarly as for the training set for the task 1, a graph that shows relationship between the number of missing values and number of examples that have the number of missing values was used, see Figure 21.

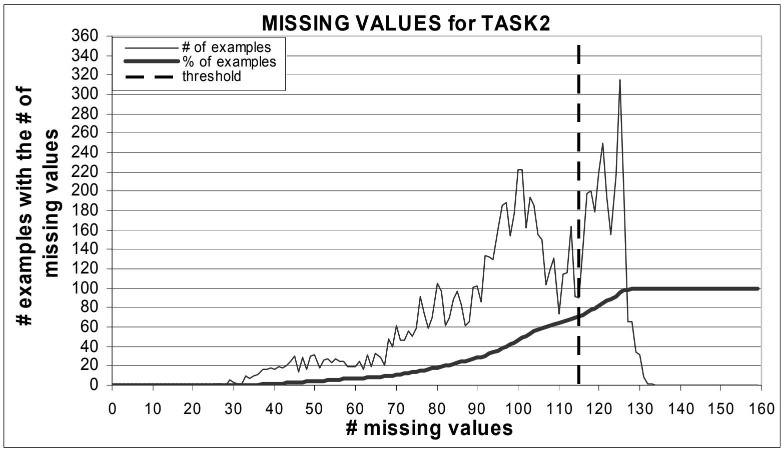


Figure 21. The relationship showing number of examples with particular number of missing values for task 2

Similarly as for the training set for the task 1, based on the analysis of the graph, the examples with more than 115 missing values were removed. Total of 2,472 examples were removed.

After applying the above preprocessing steps, the training set for task 2 consists of 6,022 examples described by 160 attributes, and with total number of values of 963,520. The number of missing values is 541,860, which constitutes 56.2 % of the entire set.

5.4.6 Refining the Project

The described above preprocessing steps were developed in highly iterative manner, which is a common practice in case of DMKD projects. Following, a summary of the iterations, including brief description of the changes, reasons, and description of affected DMKD steps is provided.

Since the start of the project, the CF data was identified as very challenging training set. Several reasons, like large amount of missing information, incorrect records, large number of attributes, and structure of the CF data, were identified as sources of possible difficulties. The project was performed slowly, with several iterations, and careful revisions of the performed work. Four formal meetings were held to evaluate the progress and direct the research. Also, numerous other informal meeting and discussion have taken place. The results of these meeting are summarized in Table 30, which shows all major iterations during the CF project.

Table 30. Summary of the refinements performed during the analysis of the CF data

current DMKD step	returned to	reasons for modifications	summary of modifications
Data Mining	Preparation of the Data	incomplete and difficult to evaluate results	modification of the join operation, hand-coded discretization of several attributes, removal of several irrelevant attributes, redesign of the "CF pace (cf)" attribute, new format of displaying the results
Evaluation of the Discovered Knowledge	Understanding of the Data	unsatisfactory results	New data table, which describes weight and height percentiles was added, modification of the join operation to accommodate for the new table, hand-coded discretization of several attributes that were discretized automatically, deletion of examples with high number of missing values, removal of several irrelevant attributes
Evaluation of the Discovered Knowledge	Data Mining	invalidated results	10 fold cross validation test procedures, improvement in the new format of displaying the results, removed minor data inconsistencies

The above iterations represent only the major modifications performed during the project. They were performed at different steps of the DMKD process and resulted in the refinement of the process by returning, modifying, and repeating some of the previously performed steps. The redesign of the approach was guided by both the medical the DM personnel. The main reason for performing these refinements was unsatisfactory quality of results. The performed modification resulted in improving the quality of the training sets, and improving quality of representation of the results. It is very important to note that all of the performed refinements resulted in substantial improvement of the quality of the approach. The above work provides a strong validation of the iterative nature of DMKD projects.

5.5 Data Mining

The goal of the DM step is to generate new and useful knowledge from the data prepared in the preceding steps. In case of the CF data, the DM step consisted in two separate tasks. Task 1 was to discover important factors related to predefined paces of the development of the disease. Task 2 was to discover important factors related to predefined types of the CF. During the Preparation of the Data step, two training sets, one per each task, were prepared. Summary information about the sets is shown in Table 31.

Table 31. Summary of training sets for the CF project

set	size	# classes	# attrib.	test data	% missing values	% inconsistent examples	# subsets
CF1	5448	4	160	10CV	56.6	0	5
CF2	6022	3	160	10CV	56.2	0	5

The DM task for both training sets is very difficult because of the two following observations:

- both training sets are characterized by very large number of missing values. For both datasets, all examples contain some missing values, and total number of missing information is larger than the amount of complete information.
- both training sets are described by a large number of attributes, which constitute about half of the original attributes from the raw CF data. Because of the very large number of attributes, it can be expected that

the data is very specific, i.e. there are very little common patterns between different patients. Thus generated rules may be long.

The first factor was overcome by application of the MetaSqueezer system. The system is proven to be missing values resistant. The results generated by the system for the CF data again prove its ability to cope with data containing large amount of missing information.

The second factor was overcome by development of an alternative knowledge representation. The rules, generated by the MetaSqueezer, were transformed into tables, called rule and selector ranking tables, which show association of particular attributes and selectors with particular class values and within particular training subsets. These tables are computed from the rules, and provide compact and very easy to understand summary of information contained in the generated rules sets.

5.5.1 Rule and Selector Ranking Tables

The tables are generated from rules by adopting procedure proposed by (Cios and Kurgan, 2002a). Background information and description of the procedure is given below.

The attribute and selector ranking tables are used to rank attributes and selectors by assigning to them a goodness value that quantifies relevance of the attributes to a particular learning task. The building of the tables is based on computing goodness of each attribute and selector using classification rules generated by the MetaSqueezer system or the DataSqueezer algorithm. The attribute and selector goodness is computed in three steps (Cios and Kurgan, 2002a):

1. Each rule generated by the MetaSqueezer system or DataSqueezer algorithm consists of multiple selectors and has assigned goodness value equal to the number of positive examples it covers. The goodness is computed during rule generation. Its value is converted into percentage of the size of positive training data.
2. Each selector is assigned a goodness value equal to the goodness of the rule it comes from. Goodness of the same selectors from different rules is summed up, and then scaled to the (0,100) range. 100 is assigned to the highest summed value, and the remaining summed values are scaled accordingly. Scaling of the goodness values is necessary because the summed goodness for a particular selector can have value over 100, and only its ratio to other selector goodness is important.
3. For each attribute the sum of scaled goodness for all its selectors is computed and divided by the number of attribute values to obtain the goodness of the attribute.

The following example shows how to compute attribute and selector tables for data shown in Table 5. The DataSqueezer generated these two rules for class home: “IF temperature = normal THEN home” (goodness 3), and “IF temperature = low AND heart blood flow = normal THEN home” (goodness 2). The goodness of rules is converted in the following manner: rule 1 describes 3 out of 5 examples from class home. Thus, its goodness is 3/5= 60%. Rule 2 describes 2 out of 5 examples from class home, so its goodness is 2/5= 40%. Below the list of all selectors and their goodness values, computed as a sum of goodness of rules that use the selector is given:

((temperature, normal); goodness 60), ((temperature, low); goodness 40), ((temperature, high); goodness 0)

((heart blood flow, normal); goodness 40), ((heart blood flow, low); goodness 0), ((heart blood flow, high); goodness 0)

((chest pain type, 1); goodness 0), ((chest pain type, 2); goodness 0), ((chest pain type, 3); goodness 0) ((chest pain type, 4); goodness 0)

After scaling the selector goodness values to the [0-100] range, the updated goodness values are as follows:

((temperature, normal); goodness 100), ((temperature, low); goodness 66.7), ((temperature, high); goodness 0)

((heart blood flow, normal); goodness 66.7), ((heart blood flow, low); goodness 0), ((heart blood flow, high); goodness 0)

((chest pain type, 1); goodness 0), ((chest pain type, 2); goodness 0), ((chest pain type, 3); goodness 0) ((chest pain type, 4); goodness 0)

For attribute “temperature” we have the following selectors and their goodness values: ((temperature, normal); goodness 100), ((temperature, low); goodness 66.7), ((temperature, high); goodness 0)

That gives the goodness of the attribute as (100+66.7+0)/3 = 55.6. Similarly, the goodness of the "heart blood flow" attribute is 22.2 and goodness of the "chest pain type" attribute is 0. The attribute and selector goodness for the rules describing class home are as follows:

```
ATTRIBUTE : temperature (55.60  goodness)
values: normal (100.00), low (66.70), high (0.00),
ATTRIBUTE : heart blood flow (22.20  goodness)
values: normal (66.70), low (0.00), high (0.00),
ATTRIBUTE : chest pain type (0.00  goodness)
values: 1 (0.00), 2 (0.00), 3 (0.00), 4 (0.00),
```

The DataSqueezer generates the following rule for class treatment: "IF chest pain type = 4 THEN treatment" (goodness 3). Following the same computations, the attribute and selector goodness for the rules describing class treatment are:

```
ATTRIBUTE : chest pain type (25.00  goodness)
values: 1 (0.00), 2 (0.00), 3 (0.00), 4 (100.00),
ATTRIBUTE : heart blood flow (0.00  goodness)
values: normal (0.00), low (0.00), high (0.00),
ATTRIBUTE : temperature (0.00  goodness)
values: normal (0.00), low (0.00), high (0.00),
```

Next, the computed goodness values are used to generate the attribute and selector ranking tables. The table generated for the above example is shown in Table 32.

Table 32. Attribute and selector ranking table for the example data

attribute	value	class home	class treatment
temperature		55.6	0
temperature	normal	100	0
temperature	low	66.7	0
temperature	high	0	0
heart blood flow		22.2	0
heart blood flow	normal	66.7	0
heart blood flow	low	0	0
heart blood flow	high	0	0
chest pain type		0	25
chest pain type	1	0	0
chest pain type	2	0	0
chest pain type	3	0	0
chest pain type	4	0	100

The table is used to analyze the generated rules in terms of finding important attributes and selectors that were found to be correlated with a particular class. The main reason for generation of such tables is simplicity of their analysis. Instead of analyzing many possible rules, the user is shown a simple table that lists the degree of association between an attribute or a selector and a desired class. The degree of association is computed from the rules, and thus is validated by the procedures used to validate the rules. The table greatly simplifies the task of understanding and analysis of the results. Once the user finds an interesting correlation, all rules that were used to compute it can be pulled and displayed.

After developing the tables and using them in the CF project, one more modification was proposed and implemented. To make the analysis of the table easier, the values representing the degree of association were color-coded. Three thresholds were designed: for value of 0 the corresponding cell in the table is white, for values from (0, 50) interval the color is gray, and for values from [50, 100] interval the color is black. The white color shows no association relationship. The gray color shows an association, while the black color shows a strong association. This greatly simplified the analysis of the table making it a very user-friendly and easy to carry-out task. As an example, the above table in color coded version is shown in Table 33.

Table 33. Color coded attribute and selector ranking table for the example data

attribute	value	class home	class treatment
temperature		■	
temperature	normal	■	
temperature	low	■	
temperature	high		
heart blood flow		▒	
heart blood flow	normal	■	
heart blood flow	low		
heart blood flow	high		
chest pain type			▒
chest pain type	1		
chest pain type	2		
chest pain type	3		
chest pain type	4		■

To even further improve the simplicity of the table, the attributes and selectors that do not exhibit association with any of the classes may be removed from it. Table 34 show the attribute and selector ranking table with removed attributes and selectors.

Table 34. Color coded attribute and selector ranking table with removed irrelevant attributes for the example data

attribute	value	class home	class treatment
temperature		■	
temperature	normal	■	
temperature	low	■	
heart blood flow		▒	
heart blood flow	normal	■	
chest pain type			▒
chest pain type	4		■

The attribute and selector tables are generated from rules generated by the MetaSqueezer system. A separate table is generated from rules generated by the DataSqueezer for each of the training subsets, called *sub-table$_i$*, where *i* is a subset index. Another table is generated for the meta-rules generated in the meta mining step of the system; called *meta-table*. The attribute and selectors tables are generated in three steps. First all *sub-table$_i$* tables are merged into a single attribute and selectors table, where entries for each of the classes are subdivided into specific intervals. Next, the *meta-table* is used to adjust the values in the existing attribute and selector table. If a specific entry exists in both tables, the value in the attribute and selector table will be summed with the value from the *meta-table*. Otherwise it is left unchanged. Finally, the values in the attribute and selector table are scaled to the (0,100) range. The table shows the associations between each of the attributes and selectors used in the rules, and the class attribute, separately for each input subset. An example table, generated from rules generated for the CF data, is shown later in the chapter.

An example attribute and selector ranking table resulting from manipulating a *sub-table$_1$*, a *sub-table$_2$*, and a *meta-table* is shown in Figure 22.

attribute	value	class home	class treatment
temperature		■	
temperature	normal	■	
temperature	low	■	
heart blood flow		▒	
heart blood flow	normal	■	
chest pain type			▒
chest pain type	4		■

a)

attribute	value	class home	class treatment
temperature		▒	▒
temperature	normal	■	
heart blood flow		▒	▒
heart blood flow	normal	■	
chest pain type			▒
chest pain type	3	▒	▒
chest pain type	4		▒

b)

attribute	value	class home	class treatment
temperature	normal	■	
heart blood flow		▒	
heart blood flow	normal	■	
chest pain type			▒
chest pain type	4		■

c)

attribute	value	class home		class treatment	
		I1	I2	I1	I2
temperature		■	▒		▒
temperature	normal	■	■		
temperature	low	▒			
heart blood flow		■	■		▒
heart blood flow	normal	■	■		
chest pain type				■	■
chest pain type	3		▒		▒
chest pain type	4			■	■

d)

Figure 22. a) *sub-table$_1$* generated for data from interval 1 (I1), b) *sub-table$_2$* generated for data from interval 2 (I2), c) *meta-table* d) attribute and selector ranking table.

The main use of the attribute and selector ranking table is to enable human evaluation of the generated rules by domain experts. Since often they have limited time to perform the evaluations any method, like the one just described, that simplifies the task is always welcome. Analysis of attribute and selector ranking tables is performed by the following procedure:

- analysis of the class attribute to generate a list of values that can be grouped based on similarities inferred from available background knowledge. This list, called the class list, includes each of the values separately, but may include some grouping of values. This step concerns analysis of values from the first row in the attribute and selector ranking table. In the example table in Figure 22d, the class list consists of {(home), (treatment)},

- finding significant attributes and selectors described in the attribute and selector ranking table. Each of the rows of the table, except the first, describes degree of association of a particular attribute or selector and class values described by the shaded areas in columns. For each attribute we find if grey or black areas that represent association of an attribute with all classes are related to any of the elements from the class list. If they are related to one element from the list then add the attribute or selector to the list of, possibly, significant selectors and attributes. This list contains an attribute or selector name along with the names of classes from the class list, to which the attribute exhibits association together with information about the degree of this association, and information for which intervals it was exhibited. In case of the example table in Figure 22d, the list of significant selectors and attributes consists of the following five entries: {(temperature normal, strong association with class home in I1 and I2), (temperature low, association with class home in I1), (heart blood flow normal, strong association with class home in I1 and I2), (chest paint type, strong association with class treatment in I1 and I2), (chest pain type 4, strong association with class treatment in I1 and I2)}.

- evaluation of findings from the list of the significant attributes and selectors. This is usually performed by domain experts who decide validity and usefulness of each of the generated associations.

The attribute and selector ranking tables generated from the training data for both tasks in the CF project, along with evaluation performed by medical professionals are described in section 5.6.

5.5.2 Experimental Results

Both training sets were used as an input to the MetaSqueezer system. First, the system was used to generate a set of production rules. Next, the rules were evaluated, in terms of performing two sets of tests for each of the defined tasks:

- first test generates a set of rules from the entire training set. The rules are used to generate the attribute and selector ranking tables, and tested on the same training set. The results report accuracy on the training data, number of generated rules and selectors. This set is used to provide results that are analyzed by the clinicians.
- second test takes the training set and performs 10 fold cross validation with the system with the same setting as for the first test. The results report verification test results, running time, and number of rules and selectors for each of the ten runs, and their mean values. The results of the second test provide reliable analysis of the simplicity, accuracy,

efficiency, and flexibility of the system, but could not be used by the clinicians since ten different rule sets are generated. Instead, this set is used to provide validation of results generated for the first set since both test are performed for the same system settings.

Table 35 shows results achieved by the MetaSqueezer system for the training set for task 1 and when performing 10 cross validation test. The table shows accuracy, specificity, sensitivity, running time, and number of rules and selectors achieved in each of the ten runs, and their corresponding mean and standard deviation values.

Table 35. The 10 fold cross validation results for task 1

Trial	1	2	3	4	5	6	7	8	9	10	*StDev*	Mean
Accuracy	57.8	58.2	65.1	60.6	59.1	59.1	66.8	60.4	61.7	61.5	*2.94*	**61.0**
Specificity	88.9	89.3	90.2	88.8	90	89.6	91.2	90.5	91	89.8	*0.8*	**89.9**
Sensitivity	57	59.2	60.1	63.3	60.7	52.6	63.4	58.4	59	56.6	*3.22*	**59.0**
Time [msec]	8663	8501	8169	8537	8428	8794	8663	8360	8413	9010	*239*	**1min 25sec 53msec**
# Rules	452	448	428	494	493	345	470	435	356	434	*50.5*	**436**
# Selector	4437	4320	4125	4710	4720	3393	4603	4213	3460	4196	*467*	**4.22E+03**

The results show that the system generates accurate rules. Two factors need to be considered to evaluate accuracy of the system. First, the data contain only about 44% of complete information. Second, the default hypothesis, where the most frequent class is selected, for that training set has 34.2% accuracy. Thus the rules generated by the MetaSqueezer system are accurate, since they achieve 61% accuracy for the 10 CV tests, which is significantly better than the default hypothesis, and they are generated from data containing high amount of missing information. Also, the rules achieve high and comparable with accuracy values of sensitivity and specificity, which further proves their high quality. The simplicity of results generated by the system is also high. The system generates only 436 rules for almost 6000 examples described by 160 attributes. The average number of selectors per rule, which is 9.7, is very low. The system generates the rules in about 86 seconds, which is a very good result considering the size of the training set. Thus, the system is also characterized by high efficiency. The system is also highly flexible since it generates accurate and simple results for input data that contain all attribute types, and is characterized by high number of missing values.

The summary of results achieved when generating rules from the entire training set for the task 1 is shown in Table 36. The table compares the results with the results achieved during the 10 CV tests.

Table 36. The summary of test results for task 1

Test type	accuracy	# rules	# selectors
Using training set	**67.6**	**498**	**4838**
10 CV, mean values	**61.0**	**436**	**4220**

The results show that the MetaSqueezer generates slightly more accurate rules when using the entire training set as the input. This is a common result, which shows that accuracy of results can be increased when using more data. The results show that the rules generated by the system can be trusted as a source of useful and reliable information about the patterns which are associated with different paces of CF.

Table 37 shows the results achieved by the MetaSqueezer system for the training set for task 2 and when performing 10 cross validation test. Similarly as for task 1, the table shows accuracy, specificity, sensitivity, running time, and number of rules and selectors achieved in each of the ten runs, and their corresponding mean and standard deviation values.

Table 37. The 10 fold cross validation results for task 2

Trial	1	2	3	4	5	6	7	8	9	10	*StDev*	Mean
Accuracy	57.4	54.1	69.8	41.3	53.6	51.1	73.8	71	69.3	73.1	*11.3*	**61.4**
Specificity	88.6	86.7	89.8	88.3	85.5	91.2	92.2	88.7	89	90.1	*1.97*	**89**
Sensitivity	71	67.5	66.4	54.6	64.1	57.4	73.8	73.2	68	76.9	*7.11*	**67.3**
Time [msec]	16918	16289	17865	15709	16626	15720	17550	17232	17765	17577	*811*	**2min 49sec 25msec**
# Rules	498	490	804	215	502	204	791	770	758	749	*233*	**578**
# Selector	5211	5073	8387	2238	5240	2039	8307	8093	7916	7648	*2.44E+3*	**6.02E+3**

The results show that the system generates accurate rules. Again, we note that the data contain only about 44% of complete information, and the default hypothesis for that training set has 46.0% accuracy. Thus the rules generated by the MetaSqueezer system are accurate, sine they achieve 61% accuracy for the 10 CV tests, which is significantly better than the default hypothesis, and they are generated from data containing high amount of missing information. The rules achieve also high and comparable with accuracy values of sensitivity and specificity, which provides additional validation of their high quality. The simplicity of results generated by the system is also high. The system generates only 578 rules for almost 6000 examples described by 160 attributes. Thus, the average number of selectors per rule, which is 10.4, is very low. The system generates the rules in about 170 seconds, and thus is also characterized by high efficiency. Finally, the system is highly flexible since it generates accurate and simple results for input data that contain all attribute types, and is characterized by high number of missing values. The results achieved by the MetaSqueezer for the training data describing task 2 are comparable, in terms of quality, to the results achieved for the task 1.

The summary of results achieved when generating rules from the entire training set for the task 2 is shown in Table 38. The table compares the results with the results achieved during the 10 CV tests.

Table 38. The summary of test results for task 2

Test type	accuracy	# rules	# selectors
Using training set	**77.2**	**790**	**4809**
10 CV, mean values	**61.4**	**578**	**6020**

Similarly as for task 1, the results show that the MetaSqueezer generates more accurate rules when using the entire training set as the input. The results show that the rules generated by the system can be trusted as a source of useful and reliable information about the patterns which are associated with different kinds of CF.

The summary of results achieved by the MetaSqueezer system for both tasks is shown in Table 39.

Table 39. Summary of the benchmarking tests for the MetaSqueezer system

task	accuracy	simplicity	efficiency	flexibility
1	high	high	very high	highly flexible
2	high	high	high	highly flexible

5.6 Evaluation of the Discovered Knowledge

The DM step generated two sets of rules, one set for each of the defined tasks. The rules were transformed into the rule and selector tables, which were presented to the domain experts.

The tables were analyzed by using the following 4 grade scale:

- 1+ was assigned for trivial, not useful results. By default such mark was simply omitted from being displayed on the table. The associations described by that mark are considered not useful from the medical perspective.
- 2+ was assigned for results that are of little interest. The associations described by that mark are considered of marginal value from the medical perspective.
- 3+ was assigned for interesting, but already known results. The associations described by that mark are considered interesting, but were already discovered by other researchers. Such results are used to provide validation of the results generated by the system.
- 4+ was assigned for very interesting, and unknown results. The associations described by that mark are of the highest value, since they

show very important finding which are not yet confirmed or reported in the professional literature. Such findings, if found, are the basis for evaluating the entire project as successful. Also, they directly lead to high quality publication in a known medical journal.

The evaluation was performed by Dr Frank Accurso, who is a cystic fibrosis expert, director of the CF care center, with background in pediatric pulmonology. The analysis of the results was performed manually based on the attribute and selector ranking tables for the two tasks, using the procedure described in 5.5.1.

The table describing results for the first task, shown in Figure 23, was analyzed in these steps:

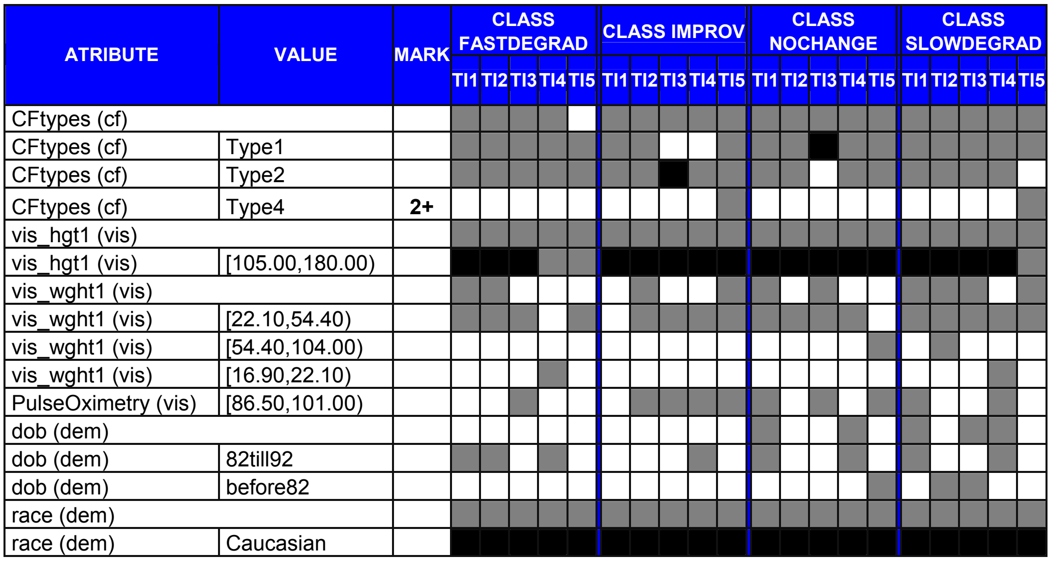
- the class list consists of {(improv), (nochange), (slowdegrad), (fastdegrad), (nochange, improve), (improve, slowdegrad), (slowdegrad, fastdegrad), (nochange, slowdegrad, fastdegrad)}. It includes elements that consist of tuples of class values; the groupings were considered as valuable by the domain experts.
- the list of significant selectors and attributes is shown below

attribute or selector	degree of association	classes	intervals
CFtypes Type4	association	slowdegrad	TI5
vis_wght1 [54.40, 104.00)	association association	nochange slodegrad	TI5 TI2
vis_wght1 [54.40, 104.00)	association association	fastdegrad slowdegrad	TI3 TI4
dob	association association	nochange slowdegrad	TI1, TI4 TI1, TI3, TI4
dob before1982	association association	nochange slowdegrad	TI5 TI2, TI3
race Indian	association	slowdegrad	TI5
race Black	association	improv	TI1, TI3, TI4
race Asian	association	slowdegrad	TI4

group C	association association association	nochange slowdegrad fastdegrad	TI4 TI1, TI2 TI1, TI2, TI4
group NSB	association	improve no change slowdegrad fastdegrad	TI1, TI2, TI3, TI4, TI5 TI1, TI2, TI3, TI4 TI1, TI4 TI1
group MI	association	improv	TI1, TI2
group FN	association	improv	TI2, TI3
motage	association	nochange slowdegrad	TI1, TI2, TI4 TI4
motage [22.50, 48.50)	association	improve no change slowdegrad fastdegrad	TI1, TI4, TI5 TI1, TI2, TI3, TI4, TI5 TI1, TI4 TI1, TI4
motage [19.50, 22.50)	association	slowdegrad fastdegrad	TI2 TI3
birthord 4	association	slowdegrad fastdegrad	TI1, TI2 TI4
numsib 3	association	nochange slowdegrad	TI2 TI4
numsib 4	association	fastdegrad	TI1, TI2, TI3
cfsib 1	association	slowdegrad fastdegrad	TI1, TI2, TI4 TI4
deltype cspln	association	slowdegrad fastdegrad	TI1, TI4 TI1
deltype unk	association	improv nochange	TI2, TI3, TI4 TI1, TI2, TI4
deltype vagbr	association	improv	TI4
mecil	association	improv nochange	TI4 TI1, TI4
mecil unk	association	improv	TI5
mecil TreatedSurgically	association	fastdegrad	TI4
irt	association	slowdegrad	TI4
irt No	association	fastdegrad slowdegrad nochange	TI1, TI2 TI2, TI3 TI5
irt Yes	association	nochange slowdegrad	TI4 TI4
FalseNeg	association	slowdegrad	TI4
FalseNeg -	association	slowdegrad	TI2, TI3
FalseNeg No	association	slowdegrad	TI4
Clinic Transplant	association	fastdegrad	TI5
age@irt1	association	nochange	TI1, TI2
irt2 [324.00, 826.00)	association	nochange slowdegrad	TI4 TI1
age@irt2 [-0.50, 27.50)	association	fastdegrad	TI2
irt3	association	improv	TI3
sweatna1 [44.5, 50.00)	association	slowdegrad	TI2, TI3
sweatk1 [24.50, 46.00)	association	improv nochange	TI3, TI4, TI5 TI1, TI2, TI4
sweatcl1 [-11.00, 95.50)	association	improv	TI1, TI2, TI4
sweatna2	association	nochange slowdegrad	TI1, TI2, TI3 TI3
sweatk2	association	nochange	TI2
sweatk2 [19.50, 58.50)	association	nochange slowdegrad	TI1, TI2, TI4 TI1, TI4

sweatcl2	association	nochange	TI2, TI3, TI4
sweatna3 [75.00, 109.00)	association	improv nochange	TI3 TI1, TI3
sweatk3 [5.50, 26.50)	association	improv nochange	TI3 TI1
sweatcl3 [99.50, 113.00)	association	nochange	TI3
sweatk4 [10.50, 39.50)	association	improv	TI3
tobraresistent? Suscept.	association	slowdegrad	TI3
tobraresistent? NotDone	association	improv	TI5
ciproresistant? Suscept.	association	fastdegrad	TI3
na [129.00, 143.00)	association	slowdegrad	TI4
prot [-1.95, 7.95)	association	slowdegrad	TI4
vita [0.44, 748.00)	association	slowdegrad	TI4
wbc [4.05, 18.00)	association	slowdegrad	TI4
hct [27.40, 45.50)	association	slowdegrad	TI4
mch [24.90, 91.40)	association	slowdegrad	TI4
machc [30.40, 35.80)	association	slowdegrad	TI4
rdw [-0.85, 15.40)	association	slowdegrad	TI4
HAZ [-2.91, -1.87)	association	fastdegrad	TI1
WZH [-1.80, 1.51)	association	fastdegrad slowdegrad	TI3 TI4
age@diagnosis 025till2	association	fastdegrad slowdegrad	TI2, TI3, TI5 TI2, TI3
age@diagnosis above7	association	nochange	TI5

- evaluation of the findings from the list of significant attributes and selectors is shown, by marks, in Figure 23, which were assigned by the experts based on their expertise of the domain. The marks were proposed and assigned by Dr. Accurso and Marci Sontag.



ATRIBUTE	VALUE	MARK	CLASS FASTDEGRAD					CLASS IMPROV					CLASS NOCHANGE					CLASS SLOWDEGRAD				
			TI1	TI2	TI3	TI4	TI5	TI1	TI2	TI3	TI4	TI5	TI1	TI2	TI3	TI4	TI5	TI1	TI2	TI3	TI4	TI5
CFtypes (cf)			▨	▨	▨	▨	□	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
CFtypes (cf)	Type1		▨	▨	▨	▨	▨	▨	▨	□	□	▨	▨	▨	■	▨	▨	▨	▨	▨	▨	▨
CFtypes (cf)	Type2		▨	▨	▨	▨	▨	▨	▨	■	▨	▨	▨	▨	□	▨	▨	▨	▨	▨	▨	□
CFtypes (cf)	Type4	2+	□	□	□	□	□	□	□	□	□	▨	□	□	□	□	□	□	□	□	□	▨
vis_hgt1 (vis)			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
vis_hgt1 (vis)	[105.00,180.00)		■	■	■	▨	▨	■	■	■	■	■	■	■	■	■	■	■	■	■	■	▨
vis_wght1 (vis)			▨	▨	□	□	□	□	▨	□	□	▨	▨	□	▨	□	□	▨	▨	▨	□	▨
vis_wght1 (vis)	[22.10,54.40)		▨	▨	▨	□	▨	□	▨	▨	▨	▨	▨	▨	▨	▨	□	▨	▨	▨	▨	▨
vis_wght1 (vis)	[54.40,104.00)		□	□	□	□	□	□	□	□	□	□	□	□	□	□	▨	□	▨	□	□	□
vis_wght1 (vis)	[16.90,22.10)		□	□	□	▨	□	□	□	□	□	□	□	□	□	□	□	□	□	□	▨	□
PulseOximetry (vis)	[86.50,101.00)		□	□	▨	□	□	□	▨	▨	▨	▨	▨	□	▨	□	▨	▨	□	□	▨	□
dob (dem)			□	□	□	□	□	□	□	□	□	□	▨	□	□	▨	□	▨	□	▨	▨	□
dob (dem)	82till92		▨	▨	□	▨	□	□	□	□	▨	□	▨	□	□	▨	□	▨	□	□	▨	□
dob (dem)	before82		□	□	□	□	□	□	□	□	□	□	□	□	□	□	▨	□	▨	▨	□	□
race (dem)			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
race (dem)	Caucasian		■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■

race (dem)	Indian																					▨
race (dem)	Black	**3+**						▨		▨	▨											
race (dem)	Asian																				▨	
group (dem)			▨					▨	▨	▨	▨		▨	▨		▨		▨			▨	
group (dem)	C	**3+/4+**	▨	▨		▨										▨		▨	▨			
group (dem)	NBS	**3+/4+**	▨					▨	▨	▨	▨	▨	▨	▨	▨	▨		▨			■	
group (dem)	MI	**3+/4+**						▨	▨													
group (dem)	FN	**3+/4+**							▨	▨												
marital (dem)			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨		▨	▨	▨	▨	▨	▨	▨	
marital (dem)	Married		▨	▨	▨	■	▨	■	■	■	■		▨	▨	▨	■	▨	▨	▨	▨	▨	
marital (dem)	Unknown											▨					▨	▨				
marital (dem)	Divorced				▨						▨	▨							▨			▨
motage (dem)													▨	▨		▨					▨	
motage (dem)	[22.50,48.50)	**3+**	▨			▨		▨			▨	▨	▨	▨	▨	▨	▨	▨			▨	
motage (dem)	[19.50,22.50)	**3+**			▨														▨			
birthord (dem)			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨		▨	▨	▨	▨	▨
birthord (dem)	3		▨			▨		▨			▨	▨	▨	▨		▨					▨	
birthord (dem)	2		▨	▨	▨	■	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	■	
birthord (dem)	1		▨	▨	▨	▨	■	▨	▨	■	■		▨	▨	▨	▨		▨	▨	▨	▨	■
birthord (dem)	4					▨												▨	▨			
numsib (dem)			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨		▨	▨
numsib (dem)	2		▨	▨	▨	▨	▨	▨			▨	▨	▨					▨	▨		▨	▨
numsib (dem)	1		▨	▨	▨	■	▨	▨	▨	▨	▨		▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
numsib (dem)	3															▨			▨			
numsib (dem)	0		▨	▨	▨	▨		▨	▨	▨	▨		▨	▨		▨		▨			▨	
numsib (dem)	4		▨	▨	▨																	
numsib (dem)	5							▨	▨	▨												
cfsib (dem)					▨	▨		▨			▨		▨		▨	▨		▨			▨	
cfsib (dem)	1					▨												▨	▨		▨	
cfsib (dem)	0		▨	▨	▨	▨		▨	▨	▨	▨	▨	▨		▨	▨	▨	▨		▨	▨	
deltype (dem)			▨			▨		▨	▨		▨		▨	▨		▨		▨	▨		▨	▨
deltype (dem)	vagnl		▨	▨	▨	■		▨	▨	▨	▨		▨	▨	▨	▨	▨	▨	▨		▨	▨
deltype (dem)	csemg		▨											▨		▨						
deltype (dem)	cspln		▨															▨			▨	
deltype (dem)	unk								▨	▨	▨		▨	▨		▨						
deltype (dem)	vagbr										▨											
mecil (dem)											▨		▨			▨						
mecil (dem)	unk											▨										
mecil (dem)	TreatedSurgically	**2+**				▨																
mecil (dem)	no					▨		▨			▨		▨	▨		▨		▨			▨	
irt (dia)																					▨	
irt (dia)	No		▨	▨													▨		▨	▨		
irt (dia)	Yes															▨					▨	
FalseNeg (dia)																					▨	
FalseNeg (dia)	-																		▨	▨		
FalseNeg (dia)	No																				▨	
Clinic (dia)			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Clinic (dia)	Billings		▨	▨				▨			▨	■	▨			▨	▨		▨	▨		▨
Clinic (dia)	Adult				▨			▨	▨	▨				▨				▨	▨	▨		▨
Clinic (dia)	Denver		■	■	■	■	■	■	▨	▨	■	▨	■	■	▨	■	▨	■	▨	▨	■	■
Clinic (dia)	Colo.Spgs		▨	▨	▨	▨	▨	▨	▨	■	▨		▨	▨	▨	▨	▨	▨	▨	▨		
Clinic (dia)	GreatFalls		▨	▨	▨	▨	▨					▨					▨	▨	▨		▨	▨
Clinic (dia)	Transplant						▨															
irt1 (dia)			▨			▨					▨							▨			▨	
irt1 (dia)	[79.50,373.00)		▨	▨		▨		▨	▨		▨		▨					▨			▨	
age@irt1 (dia)													▨	▨								
age@irt1 (dia)	[-0.50,5.50)				▨	▨		▨		▨			▨	▨	▨	▨						
irt2 (dia)	[324.00,826.00)															▨		▨				

irt2 (dia)	[87.50,322.00)		▒	▒							▒		▒								▒	
age@irt2 (dia)	[-0.50,27.50)			▒																		
irt3 (dia)	[79.00,201.00)									▒												
swetna1 (dia)			▒	▒	▒	▒		▒	▒	▒	▒		▒	▒	▒	▒		▒	▒	▒	▒	
swetna1 (dia)	[54.50,152.00)		■	■	■	■	▒	■	■	■	■	▒	■	■	■	▒	▒	■	▒	▒	■	
swetna1 (dia)	[44.50,50.00)																		▒	▒		
swetk1 (dia)				▒		▒			▒		▒											
swetk1 (dia)	[24.50,46.00)	**4+**								▒	▒	▒	▒	▒		▒						
swetk1 (dia)	[16.50,24.50)		▒	▒		▒		▒	▒				▒					▒	▒		▒	
swetcl1 (dia)			▒	▒		▒		▒	▒	▒	▒	▒			▒	▒		▒	▒	▒	▒	▒
swetcl1 (dia)	[101.00,157.00)		▒	▒	▒	■	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒
swetcl1 (dia)	[-11.00,95.50)	**3+**						▒	▒		▒											
swetna2 (dia)													▒	▒	▒					▒		
swetna2 (dia)	[22.00,101.00)		▒							▒	▒		▒	■	▒	▒		▒	▒	▒	▒	
swetk2 (dia)														▒								
swetk2 (dia)	[19.50,58.50)												▒	▒		▒		▒			▒	
swetk2 (dia)	[11.50,19.50)								▒											▒		
swetcl2 (dia)														▒	▒	▒						
swetcl2 (dia)	[108.00,123.00)									▒									▒	▒		
swetcl2 (dia)	[26.50,108.00)		▒										▒	▒	▒	▒		▒			▒	
swetna3 (dia)	[75.00,109.00)									▒			▒		▒							
swetk3 (dia)	[5.50,26.50)									▒			▒									
swetcl3 (dia)	[99.50,113.00)														▒							
swetk4 (dia)	[10.50,39.50)									▒												
SOURCE (cul)							▒					▒					▒					
SOURCE (cul)	Sputum		▒	▒	▒		■	▒	▒	▒		■		▒	▒	▒	■	▒	▒	■	▒	▒
SOURCE (cul)	ThroatCulture		▒	▒		▒		▒	▒	▒	▒		▒			▒		▒	▒	▒	▒	
tobraresistent? (cul)							▒			▒		▒					▒	▒	▒	▒	▒	▒
tobraresistent? (cul)	Suscept.	**2+**																		▒		
tobraresistent? (cul)	NotDone											▒										
tobraresistent? (cul)	No		▒	▒	▒	▒	▒	▒	▒	▒	▒	■	▒	▒	▒	▒	▒	▒	▒	■	▒	▒
ciproresistant? (cul)			▒		▒		▒			▒		▒			▒		▒	▒	▒	▒	▒	▒
ciproresistant? (cul)	Suscept.				▒																	
ciproresistant? (cul)	No		▒	▒	▒	▒	▒	▒	▒	▒	▒	■	▒	▒	▒	▒	▒	▒	▒	■	▒	▒
meropenemresistant? (cul)							▒			▒		▒					▒	▒	▒	▒	▒	▒
meropenemresistant? (cul)	No		▒	▒	▒	▒	▒	▒	▒	▒	▒	■	▒	▒	▒	▒	■	▒	▒	■	▒	▒
na (mic)	[129.00,143.00)	**2+**																			▒	
prot (mic)	[-1.95,7.95)	**2+**																			▒	
vita (mic)	[0.44,748.00)	**2+**																			▒	
wbc (hem)	[4.05,18.00)	**2+**																			▒	
hct (hem)	[27.40,45.50)	**2+**																			▒	
mch (hem)	[24.90,91.40)	**2+**																			▒	
mchc (hem)	[30.40,35.80)	**2+**																			▒	
rdw (hem)	[-0.85,15.40)	**2+**																			▒	
WAZ (per)			▒	▒	▒	▒		▒	▒	▒	▒		▒	▒	▒	▒	▒	▒	▒	▒	▒	
WAZ (per)	[-1.93,1.51)		■	■	▒	▒		■	■	■	■		■	■	■	■	■	■	■	▒	■	▒
HAZ (per)			▒	▒	▒	▒	▒	▒	▒	▒	▒		▒	▒	▒	▒	▒	▒	▒	▒	▒	
HAZ (per)	[-2.91,-1.87)	**3+**	▒																			
HAZ (per)	[-1.87,2.50)		▒	▒	■	▒	▒	■	■	■	■		■	■	■	■	■	■	■	■	■	▒
WZH (per)	[-1.80,1.51)					▒															▒	
BASEpower (cul)			▒			▒			▒		▒					▒		▒				
BASEpower (cul)	[-1.00,2E4)		▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒
BASEpower2 (cul)			▒			▒	▒		▒		▒					▒		▒				▒
BASEpower2 (cul)	[-1.75,7.5E6)		▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒
BASEpower3 (cul)			▒	▒	▒	▒	▒		▒		▒				▒	▒	▒	▒	▒	▒		▒
BASEpower3 (cul)	[-1.75,1.1E8)		▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒
BASEpower4 (cul)			▒	▒	▒	▒	▒		▒						▒	▒	▒	▒		▒		▒

BASEpower4 (cul)	[-5E3,1.25E5)		▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
age@diagnosis (dia)			▨			▨			▨		▨		▨	▨	▨		▨	▨			▨	
age@diagnosis (dia)	025till2	**3+**		▨	▨		▨												▨	▨		
age@diagnosis (dia)	below025		▨	▨		■		▨	▨	▨	▨		▨	■	▨	▨	▨	▨			▨	
age@diagnosis (dia)	above7																▨					

Figure 23. The evaluation of results for task 1

The results generated by the MetaSqueezer system for task 1 can be broken into two parts:

- confirmatory results marked by 3+ describe relationships that were known previously, but give confidence in the correctness of the performed analysis. The findings include the following correlations and their medical interpretation:
  - black race and improvement of the disease; the finding suggests that the patients who are black may have less severe disease, possibly less severe CF mutations or other genetic modifiers,
  - C group and degradation of the disease for small values of FEV%; the finding suggests that patients who are conventionally diagnosed may have a faster decline in FEV1 during advanced stages of the disease
  - NBS groups and improvement of the disease; the finding suggests that the benefits of newborn screening may result in stable or improving lung function in childhood, which may be the result of closer follow-up in early childhood,
  - MI and FN groups and improvement of the disease for small values of FEV%; the finding suggests that improvement may be seen in

FEV1 in children who are false negative on the screen or who are diagnosed through meconium ileus after their decline in FEV1 has occurred,

- [22.50,48.50) values of motage and improvement or stable state of the disease; the finding suggests that children of mothers over the age of 22 years tend to have stable lung function,
- [19.50,22.50) values of motage and degradation of the disease for medium values of FEV%; the finding suggests that children with moderate lung disease who have young mothers (between 19.5 and 22.5 years) tend to have a decline in lung function,
- [-11.00,95.50) values of sweatcl1 and improvement of the disease; the finding suggests that children with lower sweat chloride values (<95.5) may have less severe lung disease,
- [-2.91,-1.87) values of HAZ and degradation of the disease for small values of FEV%; this finding suggests that children with height stunting and severe disease may have a rapid decline in FEV1,
- 025till2 values of age@diagnosis and degradation of the disease; the finding suggests that children who were diagnosed after the initial newborn period may have a more rapid decline in pulmonary function,

- new findings marked by 4+ describe findings that may be significant medically. The findings include the following correlation and its medical interpretation:
    - [24.50,46.00) value of sweatk1 and the improvement of the disease; the finding suggests that there is a possible significance to CF of genes that modify potassium levels.

The results show not only that the system generated accurate results for the task 1, based on 10 confirmatory findings, but also that the system discovered one significant finding concerning potassium levels in sweat.

The table describing results for the second task, Figure 24, was analyzed in as follows:

- the class list consists of {(type1), (type2), (type4)},
- the list of significant selectors and attributes is shown below.

attribute or selector	degree of association	classes	intervals
CFpace Improv	association	type4	TI1, TI2, TI4
vis_wght [79.80, 104.00)	association	type4	TI3
PulseOximetry	association	type4	TI4
dob	association	type1	TI1, TI4
dob before82	association	type2	TI3, TI5
race Indian	association	type4	TI1
race Black	association	type2	TI3
group C	association	type4	TI1, TI2, TI4
group NSB	association	type1 type2 type4	TI1, TI2, TI3, TI4 TI1, TI2, TI4 TI1, TI2, TI4
marital divorced	association	type2	TI5
marital separated	association	type4	TI4
motage	association	type4	TI2
motage [15.10, 24.50)	association	type2	TI1
birthord 4	association	type2	TI2, TI3
numsib 4	association	type1	TI2
deltype csemg	association	type2	TI2
deltype cspin	association	type4	TI4
deltype unk	association	type2	TI1, TI2, TI3, TI4
apgar1 5till8	association	type1	TI4
irt	association	type2	TI5

irt No	association	type2	TI3, TI5
irt Yes	association	type1	TI1
FalseNeg	association	type2	TI5
FalseNeg -	association	type2	TI5
Clinic Transplant	association	type2	TI5
irt1	association	type3	TI2
irt1 [391.00, 759.00)	association	type1	TI2
irt1 [48.50, 112.00)	association	type2	TI1
age@irt2 [8.50, 67.50)	association	type1	TI1, TI2, TI4
sweatna1 [47.50, 80.50)	association	type2	TI4
sweatna1 [80.50, 85.50)	association	type2	TI1, TI2
sweatna1 [6.00, 47.50)	association	type4	TI2
sweatk1	association	type4	TI4
sweatk1 [31.50, 105.00)	association	type4	TI4
sweatlcl1 [-11.00, 86.50)	association	type2	TI1, TI5
sweatlcl1 [153.00, 171.00)	association	type4	TI2
sweatna2	association	type1	TI1, TI2, TI3, TI4, TI5
sweatk2	association	type1	TI1, TI2, TI4
sweatna3 [68.50, 106.00)	association	type1	TI1
sweatk3 [15.50, 28.50)	association	type1	TI3
sweatlcl3 [98.50, 114.00)	association	type1	TI3
tobraresistent? Suscept.	association	type4	TI2, TI3
ciproresistant? Suscept.	association	type4	TI2
meropenemresistant? Suscept.	association	type4	TI3
ast [9.50, 145.00)	association	type4	TI1
WZH [-1.11, 1.31)	association	type1	TI4
age@diagnosis 2till7	association	type4	TI2

- evaluation of findings from the list of the significant attributes and selectors is shown by marks in Figure 24, assigned by Dr. Accurso and Marci Sontag.

ATRIBUTE	VALUE	MARK	CLASS TYPE1					CLASS TYPE2					CLASS TYPE4				
			TI1	TI2	TI3	TI4	TI5	TI1	TI2	TI3	TI4	TI5	TI1	TI2	TI3	TI4	TI5
CFpace (cf)			▨	▨		▨		▨	▨	▨	▨		▨	▨		▨	▨
CFpace (cf)	NoChange		▨	▨		▨		▨	▨	▨	▨		▨	▨	▨		
CFpace (cf)	FastDegrad				▨					▨	▨	▨					
CFpace (cf)	SlowDegrad									▨							▨
CFpace (cf)	Improv	3+											▨	▨		▨	
vis_hgt1 (vis)			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	
vis_hgt1 (vis)	[102.00,188.00)		■	■	■	■	■	■	■	■	■	■	■	■	■	■	▨
vis_wght1 (vis)			▨	▨		▨		▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
vis_wght1 (vis)	[15.30,53.30)		■	■	▨	■	▨	■	■	■	■	▨	■	■	▨	▨	▨
vis_wght1 (vis)	[53.30,79.80)				▨											▨	
vis_wght1 (vis)	[79.80,104.00)														▨		
PulseOximetry (vis)																▨	
PulseOximetry (vis)	[92.50,101.00)		▨			▨		▨			▨		▨	▨		▨	
dob (dem)			▨			▨											
dob (dem)	82till92		▨	▨		▨		▨	▨		▨					▨	
dob (dem)	before82									▨		▨					
race (dem)			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨

race (dem)	Caucasian		■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
race (dem)	Unknown											▒			▒		▒
race (dem)	Indian												▒				
race (dem)	Black									▒							
group (dem)				▒		▒		▒	▒		▒		▒	▒		▒	
group (dem)	C	**2+**											▒	▒		▒	
group (dem)	NBS	**2+**	▒	▒	▒	▒		▒	▒		▒		▒	▒		▒	
marital (dem)			▒	▒		▒		▒	▒	▒	▒		▒	▒	▒	▒	
marital (dem)	Married		■	▒	▒	■		▒	▒	▒	■	▒	▒	■	▒	▒	
marital (dem)	Unknown		▒						▒			▒	▒				
marital (dem)	Divorced											▒					
marital (dem)	Separated															▒	
motage (dem)														▒			
motage (dem)	[25.50,37.50)		▒	▒		▒		▒			▒		▒	▒		▒	
motage (dem)	[15.10,24.50)							▒									
birthord (dem)			▒	▒	▒	▒	▒	▒	▒	▒	▒		▒	▒		▒	▒
birthord (dem)	3		▒	▒		▒			▒	▒	▒	▒	▒	▒		▒	
birthord (dem)	2		▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	■	▒	▒	■	
birthord (dem)	1		▒	▒	▒	▒	■	▒	▒	▒	▒	▒	▒		▒		■
birthord (dem)	4								▒	▒							
numsib (dem)			▒	▒	▒	▒	▒	▒	▒	▒	▒		▒	▒	▒	▒	▒
numsib (dem)	2		▒	▒		▒	▒		▒	▒	▒	▒	▒	▒	▒	▒	■
numsib (dem)	1		▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	■	▒
numsib (dem)	0		▒	▒	▒	▒		▒	▒	▒	▒		▒			▒	
numsib (dem)	4			▒													
cfsib (dem)			▒	▒	▒	▒	▒	▒	▒		▒		▒			▒	
cfsib (dem)	1		▒	▒		▒		▒	▒	▒	▒		▒	▒			
cfsib (dem)	0		▒	▒	■	■	■	▒	▒	▒	■		■	▒	▒	■	
deltype (dem)			▒	▒		▒		▒	▒		▒		▒	▒	▒	▒	▒
deltype (dem)	vagnl		▒	▒	▒	▒		▒	▒	▒	▒	▒	▒	▒	▒	▒	■
deltype (dem)	csemg								▒								
deltype (dem)	cspln															▒	
deltype (dem)	unk							▒	▒	▒	▒						
apgar1 (dem)	5till8					▒											
mecil (dem)			▒			▒		▒			▒					▒	
mecil (dem)	no		▒	▒	▒	▒		▒	▒		▒		▒	▒		▒	
dcmot (dem)						▒		▒			▒						
dcmot (dem)	yes		▒	▒	▒	▒		▒	▒		▒		▒	▒		▒	
irt (dia)												▒					
irt (dia)	No									▒		▒					
irt (dia)	Yes		▒														
FalseNeg (dia)												▒					
FalseNeg (dia)	-											▒					
Clinic (dia)			▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒
Clinic (dia)	Billings		▒	▒	▒	▒	▒		▒			▒	▒	▒	▒	▒	
Clinic (dia)	Adult			▒	▒				▒	▒		▒					
Clinic (dia)	Denver		■	■	▒	■	■	■	▒	▒	■	▒	■	■	▒	■	■
Clinic (dia)	Colo.Spgs		▒	▒	▒	▒	▒	▒	▒	▒	▒	▒		▒			
Clinic (dia)	GreatFalls									▒		▒	▒	▒	▒		▒
Clinic (dia)	Transplant											▒					
irt1 (dia)														▒			
irt1 (dia)	[391.00,759.00)	**3+**		▒													
irt1 (dia)	[112.00,285.00)		▒			▒					▒		▒	▒		▒	
irt1 (dia)	[48.50,112.00)							▒									
age@irt1 (dia)	[-0.50,2.50)		▒	▒		▒		▒			▒						
irt2 (dia)			▒			▒		▒									
irt2 (dia)	[93.00,477.00)		▒	▒		▒		▒	▒		▒			▒		▒	
age@irt2 (dia)	[8.50,67.50)		▒	▒		▒											
swetna1 (dia)								▒			▒			▒			
swetna1 (dia)	[47.50,80.50)										▒						
swetna1 (dia)	[85.50,152.0)		▒	▒	▒	▒	▒	▒	▒	▒	▒	▒		▒		▒	

swetna1 (dia)	[80.50,85.50)							▒	▒								
swetna1 (dia)	[6.00,47.50)													▒			
swetk1 (dia)																▒	
swetk1 (dia)	[31.50,105.00)															▒	
swetk1 (dia)	[4.50,19.50)						▒				▒		▒				
swetk1 (dia)	[20.50,31.50)		▒	▒		▒		▒						▒		▒	
swetcl1 (dia)			▒			▒		▒	▒	▒			▒	▒		▒	▒
swetcl1 (dia)	[113.00,153.00)		▒							▒				▒			▒
swetcl1 (dia)	[86.50,113.00)		▒	▒	▒	▒		▒	▒	▒	▒		▒	▒		▒	
swetcl1 (dia)	[-11.00,86.50)							▒				▒					
swetcl1 (dia)	[153.00,171.00)													▒			
swetna2 (dia)			▒	▒	▒	▒	▒										
swetna2 (dia)	[60.50,139.00)		▒	▒	▒	▒	▒	▒	▒	▒	▒						
swetk2 (dia)			▒	▒		▒											
swetk2 (dia)	[13.50,32.50)		▒	▒		▒		▒	▒		▒						
swetcl2 (dia)			▒	▒		▒									▒		
swetcl2 (dia)	[85.50,153.00)		▒	▒	▒	▒		▒	▒	▒	▒				▒		
swetna3 (dia)	[68.50,106.00)		▒														
swetk3 (dia)	[15.50,28.50)				▒												
swetcl3 (dia)	[98.50,114.00)				▒												
SOURCE (cul)												▒		▒	▒		▒
SOURCE (cul)	Sputum		▒	▒	▒	▒	▒	▒	▒	▒	▒	■	▒	▒	■		■
SOURCE (cul)	ThroatCulture		▒	▒		▒		▒	▒	▒	▒	▒	▒	▒	▒	▒	
tobraresistent? (cul)							▒			▒		▒		▒	▒		▒
tobraresistent? (cul)	Suscept.													▒	▒		
tobraresistent? (cul)	NotDone					▒					▒		▒	▒		▒	
tobraresistent? (cul)	No		▒	▒	▒	▒	▒	▒	▒	▒	▒	■	▒	▒	■	▒	■
ciproresistant? (cul)				▒	▒					▒	▒	▒		▒	▒		▒
ciproresistant? (cul)	Suscept.													▒			
ciproresistant? (cul)	NotDone										▒		▒	▒			
ciproresistant? (cul)	No		▒	▒	▒	▒	▒	▒	▒	▒	▒	■	▒	▒	■	▒	■
meropenemresistant? (cul)							▒			▒		▒		▒	▒		▒
meropenemresistant? (cul)	NotDone										▒		▒	▒			
meropenemresistant? (cul)	No		▒	▒	▒	▒	▒	▒	▒	■	▒	■	▒	▒	■	▒	■
meropenemresistant? (cul)	Suscept.														▒		
ast (mic)	[9.50,145.00)												▒				
WAZ (per)			▒	▒	▒	▒		▒	▒	▒	▒	▒	▒	▒		▒	
WAZ (per)	[-1.75,1.36)		■	■	▒	■	▒	■	■	■	■	▒	■	▒		▒	
HAZ (per)			▒	▒	▒	▒		▒	▒	▒	▒	▒	▒	▒	▒	▒	
HAZ (per)	[-1.53,2.53)		■	■	▒	■	▒	■	■	■	■	■	■	▒	▒	■	
WZH (per)	[-1.11,1.31)					▒											
BASEpower (cul)			▒			▒		▒	▒		▒	▒	▒	▒	▒	▒	▒
BASEpower (cul)	[-0.50,1.15E3)		▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	■	▒	▒	▒
BASEpower2 (cul)			▒			▒		▒	▒		▒	▒	▒	▒	▒	▒	▒
BASEpower2 (cul)	[-1.75,752.00)		▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	■	▒	▒	▒
BASEpower3 (cul)			▒	▒		▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒
BASEpower3 (cul)	[-0.50,5.5E7)		▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	■	▒	▒	▒
BASEpower4 (cul)			▒	▒		▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒
BASEpower4 (cul)	[-2.5E3,2.5E3)		▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	■	▒	▒	▒
age@diagnosis (dia)			▒	▒		▒		▒	▒		▒	▒		▒		▒	
age@diagnosis (dia)	025till2					▒		▒		▒		▒		▒			
age@diagnosis (dia)	below025		▒	▒	▒	▒		▒	▒		▒		▒			▒	
age@diagnosis (dia)	2till7													▒			

Figure 24. The evaluation of results for task 2

The results generated by the MetaSqueezer system for task 2 include only several confirmatory findings. The findings include the following correlations and their medical interpretation:

- improvement of the disease and Type 4 CF; the finding suggests that Children with 2 non-F508 mutations may have mild lung disease,
- [391.00,759.00) values of irt1 and Type 1 CF for medium values of FEV%; the finding suggests that children with high IRT values at birth have moderate lung disease.

The results show that the system generates accurate results for the task 2, based on 2 confirmatory findings. No new and significant findings were discovered for task 2. The future work will include redefining classes for this task. They will include five distinct genotypes, instead of currently defined 3. This, in turn, may lead to discovery of new and significant findings that were not yet found because of too big granularity of class definitions.

As an additional validation of the usefulness of the MetaSqueezer system, as applied to the CF data, we show in Appendix D a comparison of the results it generated with the results generated when applying the DataSqueezer algorithm to the same data. The results, in terms of attribute and selector importance tables generated by the DataSqueezer algorithm show that:

- since the DataSqueezer operates on the entire data set, the tables show only associations between attributes and selectors and classes, which limits analytical capabilities of the user,

- after performing analysis of the tables, using a procedure identical to the procedure described for analysis of results generated by the MetaSqueezer system, the following was found:
    - the results for task 1 include only five 3+ (confirmatory) findings. They overlap with findings of the MetaSqueezer system. Remaining confirmatory findings and the significant finding that were discovered by the MetaSqueezer system, as well as other findings, were not present in the results by the DataSqueezer algorithm.
    - the results for task 2 include only one 3+ (confirmatory) finding. It overlaps with findings of the MetaSqueezer system. Remaining confirmatory findings that were discovered by the MetaSqueezer system, as well as other findings, were not present in the results generated by the DataSqueezer algorithm.

The above direct comparison of the results generated by the MetaSqueezer system and the DataSqueezer algorithm shows clearly the advantages of the Meta Mining system. The system is able to generate more useful results from the same data when compared with a data mining algorithm.

5.7 Using the Discovered Knowledge

The DMKD process, which was used in the project, appears to be very practical. The iterative process used for the generation of results significantly improved results generated by the MetaSqueezer system. The system generated two kinds of results: 12 confirmatory findings that prove the correctness of the system, and most importantly 1 significant new finding that shows that the system is capable of generation of useful results.

5.7.1 Summary of Results

The outcome of the project was evaluated as very successful by the domain experts. In their opinion the results constitute material that can be published in a high quality medical journal. The publication will be written upon running task 2 with new definition of classes. Even in case of not generating any new findings for that task, the new finding discovered for task 1 was evaluated as sufficient basis for the publication. The new finding discovered for task 1 was classified as possibly medically significant, and thus may help to bring new insights into this very important disease.

Chapter 6

6 Summary

In this chapter we summarize significance of our research, provide conclusions, and describe future work.

6.1 Summary and Significance

The main goal of the our research was to develop an inductive supervised ML system that is efficient, generates simple and accurate results, and is flexible to be applied to a variety of input data formats and data containing noise and missing values. As a result, a novel learning system was developed based on the meta-mining concept. To the best of our knowledge, the system is the first that applies an inductive ML algorithm within an MM setting to generate a set of production rules. The application of the MM concept resulted in achieving several desirable properties: generation of compact data models, scalability, user-friendliness in terms of the transparency of the learning process, and most importantly, simplicity of generated results.

There are several reasons why the MetaSqueezer system has proven to be successful. It satisfies all four criteria, defined in Chapter 2, which define a successful supervised IL system:

- It generates rules that result in accurate classification.

The benchmarking tests showed that the system generates rules that accurately classify test data even in the presence of noise and inconsistencies. The MetaSqueezer system achieves error rates that are within the range of error rates achieved by the most accurate IL algorithm.

- It generates simple rules

  The benchmarking tests show that the system generates small number of very compact rules. The system generates rules that on average use 2.2 selectors in the rule's descriptor, which is between 35% and 90% less than the number of selectors generated by other supervised ML algorithm.

- It generates rules very efficiently

  The theoretical complexity analysis shows that the system has linear complexity. The benchmarking tests show that the average CPU execution time of the system is 4.3 seconds, which is the best execution time among all compared inductive ML algorithms, except the DataSqueezer algorithm itself. Both results show that the system can be successfully applied to very large data sets.

- It is very flexible

The MetaSqueezer system was tested on very large data sets, both in terms of number of examples and attributes. The data sets included all types of attributes, i.e., discrete numerical, discrete nominal, continuous, and binary. They also included data sets with large amounts of noisy and missing information. Thus, the system is very flexible and robust since it generates high quality results for the diverse types of data.

In other words, the performed benchmarking tests of the system clearly show its two main advantages, namely high compactness of the generated rules and very low computational cost. These results place the MetaSqueezer system among the best inductive ML algorithms.

The usefulness of the system was also proven by applying it to a real-life project concerning analysis of CF data. The difficulty of the project was caused by the complexity of the data, high number of missing values, and our weak background knowledge about the disease. Despite that, the system generated meaningful and useful results that can be used to enhance understanding of the disease. The results generated by the system not only confirmed its correctness by "discovering" knowledge already known by the CF experts about some relationships, but also provided a new significant finding about the disease. The new finding is medically important, and concerns potassium levels in sweat. More specifically, it is related to the possible significance to CF of genes that modify potassium levels.

To summarize, the research was proven to be useful in several ways. It proposes, thoroughly tests, and applies a novel inductive supervised ML system. The results show high potential of the system. It can be used to generate non-temporal data models for supervised data, including ordered and temporal data. The ability of the system to very efficiently generate very simple and easy to understand rules makes it one of the best inductive supervised ML systems.

6.2 Future Work

The future work will include applications of the system to projects possibly concerning various domains including medicine, banking, and retail stores data. The immediate work will concentrate on a follow up on the CF project, where analysis of task 2 will be refined by redefining the class attribute.

Another task concerns investigation of alternative ways of displaying the results generated by the MetaSqueezer system. The dissertation describes one new format, in terms of attribute and selector ranking tables generated directly from the rules. The tables were used, with great success, to display and analyze results of the CF project. In the future, a study to evaluate this and other ways of displaying the generated rules will be performed. It will take into account human cognitive processes, especially information of how people assimilate new knowledge to increase the usefulness of the proposed system (Pazzani, 2000).

References

Abraham, T., and Roddick, J.F., Discovering Meta-rules in Mining Temporal and Spatio-Temporal Data, *Proceedings of the Eighth International Database Workshop*, pp. 30-41, 1997

Abraham, T., and Roddick, J. F., Incremental Meta-mining from Large Temporal Data Sets, Advances in Database Technologies, *Proceeding of the First International Workshop on Data Warehousing and Data Mining* (DWDM'98), Berlin, Germany, Lecture Notes in Computer Science. 1552, pp.41-54, 1999

Agrawal, R., and Srikant, R., Fast Algorithms for Mining Association Rules, *Proceedings of the 20th International Conference on Very Large Databases*, Santiago, Chile, 1994

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., and Verkamo, A.I., Fast Discovery of Association Rules, *Advances in Knowledge Discovery and Data Mining*, Chapter 12, AAAI/MIT Press, 1995

Blackard, J.A., *Comparison of Neural Networks and Discriminant Analysis in Predicting Forest Cover Types*, Ph.D. dissertation, Department of Forest Sciences, Colorado State University, Fort Collins, Colorado, 1998

Blake, C.L., and Merz, C.J., UCI Repository of Machine Learning Databases, http://www.ics.uci.edu/ ~mlearn/MLRepository.html, Irvine, CA: University of California, Department of Information and Computer Science, 1998

Brachman, R., and Anand, T., The Process of Knowledge Discovery in Databases: A human-centered Approach, In Fayyad, U.M., Piatesky-Shapiro, G., Smyth, P., and Uthurusamy, R., (Eds) *Advances in Knowledge Discovery and Data Mining*, AAAi/MIT Press, 1996

Bradley, P., Fayyad, U., and Reina, C., Scaling Clustering Algorithms to Large Databases, *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, Menlo Park, California, pp. 9-15, 1998

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J., *Classification and Regression Trees*, Wadsworth & Brooks, 1984

Cabena, P., Hadjinian, P., Stadler, R., Verhees, J., Zanasi, A., *Discovering Data Mining: From Concepts to Implementation*, Perentice Hall, 1998

Cai, Y., Cercone, N., and Han, J., Attribute-Oriented Induction in Relational Databases, In Piatetsky-Shapiro, G., and Frawley, W.J., *Knowledge Discovery in Databases*, AAAI Press, pp.213-228, 1991

Carbonell, J.G., Michalski R.S., and Mitchell, T., An Overview of Machine Learning, In Michalski, R.S., Carbonell J.G., and Mitchell, T.M., *Machine Learning, and Artificial Intelligence Approach*, vol. 1, Morgan-Kaufmann, pp.3-24, 1983

Catlett, J., On Changing Continuous Attributes into Ordered Discrete Attributes, *Proceedings of the European Working Session on Learning*, pp.164-178, 1991

Chan, C.C., Bartur, C., and Srinivasasn, A., Determination of Quantization Intervals in Rule Based Model for Dynamic Systems, *Proceedings of the IEEE Conference on System, Man, and Cybernetics*, Charlottesville, VA, pp.1719-1723, 1991

Ching, J.Y., Wong, A.K.C., and Chan, K.C.C., Class-Dependent Discretization for Inductive Learning from Continuous and Mixed Mode Data, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 7, pp. 641-651, 1995

Chiu, D., Wong, A., and Cheung, B., Information Discovery through Hierarchical Maximum Entropy Discretization and Synthesis, *Knowledge Discovery in Databases*, G. Piatesky-Shapiro and W.J. Frowley, ed., MIT Press, 1991

Cios, K.J. and Liu, N., An Algorithm Which Learns Multiple Covers Via Integer Linear Programming, Part I - The CLILP2 Algorithm, *Kybernetes*, vol. 24, no 2, pp. 29-50, 1995a

Cios, K.J. and Liu, N., An Algorithm Which Learns Multiple Covers Via Integer Linear Programming, Part II – Experimental Results And Conclusions, *Kybernetes*, vol. 24, no 3, pp. 24-36, 1995b

Cios, K.J., Wedding, D.K. and Liu, N, CLIP3: Cover Learning Using Integer Programming, *Kybernetes*, 26(4-5): 513-536 (The Norbert Wiener 1997 Outstanding Paper Award), 1997

Cios, K. J., Pedrycz, W., and Swiniarski, R., *Data Mining Methods for Knowledge Discovery*, Kluwer, 1998

Cios, K.J., Teresinska, A., Konieczna, S., Potocka, J., and Sharma, S., Diagnosing Myocardial Perfusion from PECT Bull's-eye Maps - A Knowledge Discovery Approach, *IEEE Engineering in Medicine and Biology Magazine*, Special issue on Medical Data Mining and Knowledge Discovery, 19(4), pp. 17-25, 2000a

Cios, K.J., Kurgan, L., Mitchell, S, Bailey, M., Duckett, D., and Gau, K., Report for the OAI Phase I Collaborative Core Research Project on Data Mining and Knowledge Discovery, the *2000 Ohio Aerospace Institute (OAI) Collaborations Forum*, Cleveland, OH, 2000b

Cios, K.J. (Ed.), *Medical Data Mining and Knowledge Discovery*, Springer, 2001

Cios, K. J. and Kurgan, L., Hybrid Inductive Machine Learning: An Overview of CLIP Algorithms. In: Jain, L.C. & Kacprzyk, J. (Eds.), *New Learning Paradigms in Soft Computing*, Physica-Verlag (Springer), pp. 276-322, 2001

Cios, K.J., and Kurgan, L., Hybrid Inductive Machine Learning Algorithm that Generates Inequality Rules, *Information Sciences*, Special Issue on Soft Computing Data Mining, accepted, 2002a

Cios, K. J., and Kurgan, L., Trends in Data Mining and Knowledge Discovery, In: Pal N.R., Jain, L.C. and Teoderesku, N. (Eds.), *Knowledge Discovery in Advanced Information Systems*, Springer, to appear, 2002b

Cios, K.J., and Moore, G.W., Uniqueness of Medical Data Mining, *Artificial Intelligence in Medicine*, 26:1-2, pp.1-24, 2002

Clark, P., and Niblett, T., The CN2 Algorithm, *Machine Learning*, 3, pp.261 283, 1989

Clark, P., and Boswell, R., Rule induction with CN2: Some Recent Improvements. *Proceedings of the Fifth European Machine Learning Conference* (EWSL-91), Berlin, Germany, pp.151-163, 1991

CRISP-DM, www.crisp-dm.org, 2001

Cystic Fibrosis Foundation, http://www.cff.org/, 2002

Cystic Fibrosis Mutation Database, http://www.genet.sickkids.on.ca/cftr/, 2003

Cystic Fibrosis Genetic Analysis Consortium, Worldwide Survey of the [Delta]F508 Mutation- Report from the Cystic Fibrosis Genetic Analysis Consortium, *American Journal of Human Genetics*, 47, pp.354-359, 1990

Data Mining Tools, http://www.rulequest.com/see5-info.html, 2002

Dietterich, T.G., and Michalski, R.S., A Comparative Review of Selected Methods for Learning from Examples, In Michalski, R.S., Carbonell J.G., and Mitchell, T.M., *Machine Learning, and Artificial Intelligence Approach*, vol. 1, Morgan-Kaufmann, pp.41-81, 1983

Dougherty, J., Kohavi, R., and Sahami, M., Supervised and Unsupervised Discretization of Continuous Features, *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 194-202, 1995

Efron, B., Computers and the Theory of Statistics, *SIAM Review*, 21, pp.460-480, 1979

Esposito, F., Malerba, D., and Semeraro, G., A Comparative Analysis of Methods for Pruning Decision Trees, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:5, pp.476-491, 1997

Esposito, F., Malerba, D., Marengo, V., Inductive Learning From Numerical And Symbolic Data: an Integrated Framework, *Intelligent Data Analysis*, 5:6, pp.445-461, 2001

Fayyad, U.M., and Irani, K.B., On the Handling of Continuous-Valued Attributes in Decision Tree Generation, *Machine Learning*, vol. 8, pp. 87-102, 1992

Fayyad, U.M., and Irani, K.B., Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning, *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, San Francisco, CA, Morgan Kaufmann, pp.1022-1027, 1993

Fayyad, U.M., Piatesky-Shapiro, G., Smyth, P., and Uthurusamy, R., *Advances in Knowledge Discovery and Data Mining*, AAAi/MIT Press, 1996a

Fayyad, U.M., Piatesky-Shapiro, G., Smyth, P., From Data Mining to Knowledge Discovery, In: *Advances in Knowledge Discovery and Data Mining*, Fayyad, U.M., Piatesky-Shapiro, G., Smyth, P., and Uthurusamy, R.(Eds.), AAAi/MIT Press, pp.1-34, 1996b

Fayyad, U.M., Piatetsky-Shapiro, G., and Smyth, P., Knowledge Discovery and Data Mining: Towards a Unifying Framework, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining* (KDD96), Portland, OR. AAAI Press, 1996c

Goebel, M., and Gruenwald, L., A Survey of Data Mining Software Tools, *SIGKDD Explorations*, 1(1), pp. 20-33, 1999

Ganti, V., Ramakrishnan, R., Gehrke, J., Powell, A.L., French, J.C., Clustering Large Data sets in Arbitrary Metric Spaces, *Proceedings of the 15th International Conference on Data Engineering* (ICDE), pp.502-511, 1999a

Ganti, V., Gehrke, J., and Ramakrishnan, R., Mining Very Large Databases, *IEEE Computer*, 32(8), pp.38-45, 1999b

Gehrke, J., Ramakrishnan, R., and Ganti, V., RainForest - a Framework for Fast Decision Tree Construction of Large Data sets, *Proceedings of the 24th International Conference on Very Large Data Bases*, San Francisco, pp. 416-427, 1998

Goldberg, D.E., *Genetics Algorithm in Search, Optimization and Machine Learning*, Addison-Wesley, 1989

Han, J., Cai, Y., and Cercone, N., Knowledge Discovery in Databases: An Attribute-Oriented Approach, *Proceedings of the 18th Conference on Very Large Databases*, pp.547-559, Canada, 1992

Hettich, S. and Bay, S. D., The UCI KDD Archive [http://kdd.ics.uci.edu]. Irvine, CA: University of California, Department of Information and Computer Science, 1999

Holland, J.H., Escaping Brittleness: the Possibilities of General Purpose Algorithms Applied to Parallel Rule-Base Systems, In Michalski, R.S., Jaime, G.C., and Mitchell, T.M. (Eds), *Machine Learning, an Artificial Intelligence Apprach*, vol. 2, Morgan Kaufmann, 1986

Holland, J.H., Holyoak, K.J., Nisbett, R.E., and Thagard, P.R., *Induction: Processes of Inference, Learning and Discovery*, Computational Models of Cognition and Perception, MIT Press, Cambridge, 1986

Holsheimer, M., and Siebes, A., *Data Mining: The Search for Knowledge in Databases*, Technical report CS-R9406, P.O. Box 94079, 1090 GB Amsterdam, 1994

Holte, R.C., Very Simple Classification Rules Perform Well on Most Commonly Used Data Sets, *Machine Learning*, 11:63-90, 1993

Huang, W., *Discretization of Continuous Attributes for Inductive Machine Learning*, Master's Thesis, Department of Computer Science, University of Toledo, Ohio, 1996

Iba, W., Wogulis, J. & Langley, P., Trading off Simplicity and Coverage in Incremental Concept Learning, *Proceedings of the Fifth International Conference on Machine Learning*, Ann Arbor, Michigan, Morgan Kaufmann, pp.73-79, 1988

Karimi, K., and Hamilton, H.J., Finding Temporal Relations: Causal Bayesian Networks vs. C4.5, *Proceedings of the Twelfth International Symposium on Methodologies for Intelligent Systems* (ISMIS'2000), Charlotte, NC, USA, pp. 266-273, 2000

Kaufman, K.A., and Michalski, R.S., Learning from Inconsistent and Noisy Data: The AQ18 Approach, *Proceedings of the Eleventh International Symposium on Methodologies for Intelligent Systems*, Warsaw, Poland, pp.411-419, 1999

Kerber, R., ChiMerge: Discretization of Numeric Attributes, *Proceedings of the Ninth International Conference on Artificial Intelligence* (AAAI-91), pp. 123-128, 1992

Kodratoff, Y., *Introduction to Machine Learning*, Morgan-Kaufmann, 1988

Kohavi, R., Scaling Up the Accuracy of Naive-Bayes Classifiers: a Decision-Tree Hybrid, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pp.202-207, 1996

Kooperberg, C., Bose, S. & Stone, C.J., Polychotomous Regression, *Journal of the American Statistical Association*, 92, pp.117-127, 1997

Kukar, M., Kononenko, I., Groselj, C., Kralj,K., and Fettich, J., Analyzing and Improving the Diagnosis of Ischemic Heart Disease with Machine Learning, *Artificial Intelligence in Medicine*, 16:1, pp.25-50, 1999

Kurgan, L.A., Cios, K.J., Tadeusiewicz, R., Ogiela, M. and Goodenday, L.S., Knowledge Discovery Approach to Automated Cardiac SPECT Diagnosis, *Artificial Intelligence in Medicine*, 23(2), pp. 149-169, 2001

Kurgan, L., and Cios, K.J., Discretization Algorithm that Uses Class-Attribute Interdependence Maximization, *Proceedings of the 2001 International Conference on Artificial Intelligence* (IC-AI 2001), Las Vegas, Nevada, pp.980-987, 2001

Kurgan, L., and Cios, K.J., DataSqueezer Algorithm that Generates Small Number of Short Rules, *IEE Proceedings: Vision, Image and Signal Processing*, submitted, 2002a

Kurgan, L., and Cios, K.J., CAIM Discretization Algorithm, *IEEE Transactions of Knowledge and Data Engineering*, accepted, 2002b

Kurgan, L., & Cios, K. J., Meta-Mining Architecture for Learning from Supervised Data, submitted, 2003a

Kurgan, L., and Cios, K.J., Fast Class-Attribute Interdependence Maximization (CAIM) Discretization Algorithm, *Proceeding of the 2003 International Conference on Machine Learning and Applications* (ICMLA'03), pp.30-36, Los Angeles, 2003b

Langley, P., *Elements of Machine Learning*, Morgan-Kaufmann, 1996

Lim, T.-S., Loh, W.-Y., and Y.-S., Shih, A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-three Old and New Classification Algorithms, *Machine Learning*, 40, pp.203-228, 2000

Linde, Y., Buzo, A., Gray, R.M., An Algorithm for Vector Quantizer Design, *IEEE Transactions on Communications*, vol. 28:1, pp. 84-95, 1980

Liu, H., and Setiono, R., Feature Selection via Discretization, *IEEE Transactions on Knowledge and Data Engineering*, vol. 9, no. 4, pp.642-645, 1997

Manilla, H., Methods and Problems in Data Mining, *Proceedings of the International Conference on Database Theory*, pp. 41-55, 1997

Matheus, C.J., Chan, P.K., and Piatesky-Schapiro, G., Systems for Knowledge Discovery in Databases, *IEEE Transactions on Knowledge and Data Engineering*, 5:6, pp.903-913, 1993

Michalski, R. S. Discovering Classification Rules Using Variable-Valued Logic System VL1, *Proceedings of the Third International Joint Conference on Artificial Intelligence*, pp.162-172, 1973

Michalski, R.S., A Theory and Methodology of Inductive Learning, In Michalski, R.S., Carbonell J.G., and Mitchell, T.M., *Machine Learning, and Artificial Intelligence Approach*, vol. 1, Morgan-Kaufmann, pp.83-143, 1983a

Michalski, R. S., A Theory and Methodology of Inductive Learning, In Michalski, R., Carbonell, J., & Mitchell, T.M. (Eds.), *Machine Learning*, Tioga Press, pp.83-129, 1983b

Michalski, R.S., Mozetic, I., Hong, J., and Lavrac, N., The Multipurpose Incremental Learning System AQ15 and Its Testing Application to Three Medical Domains, *Proceedings of the Fifth National Conference on Artificial Intelligence*, pp. 1041 1045, 1986

Mingers, J., An Empirical Comparison of Pruning Methods for Decision Tree Induction, *Machine Learning*, 4:2, pp.227-243, 1989

Mitchell T.M., *Machine Learning*, McGraw-Hill, 1997

Nillson, N.J., *Principles of Artificial Intelligence*, Spriger-Verlag, 1982

Paterson, A., and Niblett, T.B., *ACLS Manual*, Edinburgh: Intelligent Terminals, Ltd, 1987

Pazzani, M.J., Knowledge discovery from data?, *IEEE Intelligent Systems*, pp.10-13, March/April 2000

Pfahringer, B., Compression-Based Discretization of Continuous Attributes, *Proceedings of the Twelfth International Conference on Machine Learning*, pp.456-463, 1995

Piatesky-Shapiro, G., Knowledge Discovery in Real Databases: A Report on the IJCAI-89 Workshop, *AI Magazine*, 11(5), pp. 68-70, Jan. 1991

Piatetsky-Shapiro, G., and Frawley, W., (Eds), *Knowledge Discovery in Databases*, AAAI/MIT Press, 1991

Quinlan, J.R., Induction of Decision Trees, *Machine Learning*, 1, pp.81-106, 1986

Quinlan, J.R., Generating Production Rules from Decision Trees, *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pp.304-307, 1987a

Quinlan, J. R., Simplifying Decision Trees, *International Journal of Man-Machine Studies*, 27:3, pp.221-248, 1987b

Quinlan, J.R., *C4.5 Programs for Machine Learning*, Morgan Kaufmann, San Francisco, 1993

Quinlan, J.R., Improved Use of Continuous Attributes in C4.5, *Journal of Artificial Intelligence Research*, 7, pp.77-90, 1996

Rainsford, C.P., and Roddick, J.F., Adding Temporal Semantics to Association Rules, *Proceeding of the Third European Conference on Principles and Practice of Knowledge Discovery in Databases* (PKDD'99), Prague, Lecture Notes in Artificial Intelligence, 1704, pp.504-509, 1999

Rao, R.B., Lu, S.C-Y., A Knowledge-Based Equation Discovery System for Engineering Domains, *IEEE Expert*, pp.37-41, August 1993

Redman, T.C., The Impact of Poor Data Quality on the Typical Enterprise, *Communications of the ACM*, 41:2, pp.79-81, 1998

Rivest, R.L., Learning Decision Lists, *Machine Learning*, 2, pp.229-246, 1987

Roddick, J.F., and Spiliopoulou, M., A Survey of Temporal Knowledge Discovery Paradigms and Methods, *IEEE Transactions on Knowledge and Data Engineering*, 14:4, pp.750-767, 2002

Russel, S., and Norvig, P., *Artificial Intelligence, A Modern Approach*, Prentice Hall, 1995

Sacha, J.P., Cios, K.J., and Goodenday, L.S., Issues in Automating Cardiac SPECT Diagnosis. *IEEE Engineering in Medicine and Biology Magazine*, special issue on Medical Data Mining and Knowledge Discovery, 19(4), pp. 78-88, 2000

Schlimmer, J.S., *Concept Acquisition through Representational Adjustment*, (Technical Report 87-19), Ph.D. dissertation, Department of Information and Computer Science, University of California, Irvine, 1987

Shafer, J., Agrawal, R., and Mehta, M., SPRINT: A Scalable Parallel Classifier for Data Mining, *Proceedings of the 22nd International Conference on Very Large Data Bases*, San Francisco, pp. 544-555, 1996

Shank, R.C., Where is AI, *AI Magazine*, winter 1991, pp.38-49, 1991

Shannon, C.E. A Mathematical Theory of Communication, *The Bell Systems Technical Journal*, 27, pp.379-423, 1948

SPEC, Standard Performance Evaluation Corporation, http://www.spec.org/spec/, 2001

Spiliopoulou, M., and Roddick, J. F., Higher Order Mining: Modelling and Mining the Results of Knowledge Discovery, *Data Mining II – Proceedings of the Second International Conference on Data Mining Methods and Databases*, Cambridge, UK, pp.309-320, 2000

Toivonen, H., Sampling Large Databases for Association Rules, *Proceedings of the 22nd International Conference on Very Large Data Bases*, San Francisco, pp. 134-145, 1996

Tou, J.T., and Gonzalez, R.C., *Pattern Recognition Principles*, Addison-Wesley, 1974

Ullman, J.D., *Principles of Database and Knowledge-base Systems*, vol. 2, Computer Science Press, New York, 1989

Ullman, J.D., and Zaniolo, C., Deductive Databases: Achievements and Future Directions, *SIGMOD Record*, 19(4), pp.75-82, 1990

Wirth, R., and Hipp, J., CRISP-DM: Towards a Standard Process Model for Data Mining, *Proceedings of the Fourth International Conference on the Practical*

Applications of Knowledge Discovery and Data Mining, Manchester, UK, pp. 29-39, 2000

Wong, A.K.C., and Chiu, D.K.Y., Synthesizing Statistical Knowledge from Incomplete Mixed-Mode Data, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 9, pp. 796-805, 1987

Wong. A.K.C., and Liu, T.S., Typicality, Diversity and Feature Pattern of an Ensemble, *IEEE Transaction on Computers*, vol. 24, pp.158-181, 1975

Wu, X., A Bayesian Discretizer for Real-Valued Attributes, t*he Computer Journal*, vol. 39, 1996

Vlachos, P., StatLib Project Repository, http://lib.stat.cmu.edu, 2000

Zhang, T., Ramakrishnan, R., Livny, M., BIRCH: An Efficient Data Clustering Method for Very Large Databases, *Proceedings of the SIGMOD Conference*, pp.103-114, 1996

Appendix A

Abbreviations

CAIM – Class-Attribute Interdependency Maximization (algorithm), see section 3.4

CF – Cystic Fibrosis, see section 5.2

CLIP4 – Cover Learning using Inductive Programming version 4, see section 2.3.2

DM – Data Mining, see section 1.1.6

DMKD – Data Mining and Knowledge Discovery, see section 1.1.6

F-CAIM – Fast Class-Attribute Interdependency Maximization (algorithm), see section 5.4.3

KD – Knowledge Discovery, see section 1.1.6

IL – Inductive Learning, see section 1.1

ML – Machine Learning, see section 1.1.6

MM – Meta Mining, see section 3.1.1

Appendix B

Relevant Publications

The content of this dissertation is supported by several refereed articles that were published, accepted or submitted to international journals and conferences. A full list of the publications can be found at the end of this appendix. Below, an outline of the contents of each publication and their relevant appearance within the body of the dissertation are given.

The first chapter provides introduction to the dissertation. Its latter parts, especially the description of the research goals, were based on two papers that propose and describe the DataSqueezer algorithm and the MetaSqueezer system (Kurgan and Cios, 2002a; Kurgan and Cios, 2003a). Also, a book chapter was used to provide an overview of the DM and ML fields (Cios and Kurgan, 2002b).

The second chapter describes related work. The definition of qualities of inductive ML algorithms was based on (Cios and Kurgan, 2002a). The discussion of rule algorithms was supported by two papers that describe a rule algorithm called DataSqueezer (Kurgan and Cios, 2002a; Kurgan and Cios, 2003a). Similarly the discussion of hybrid algorithms was supported by two papers that describe a family of hybrid algorithms called CLIP (Cios and Kurgan, 2001; Cios and Kurgan, 2002a).

The third chapter provides an overview of the MetaSqueezer system and describes its major elements. The system's overview is based on (Kurgan and

Cios, 2003a). The description of the DataSqueezer system is based on (Kurgan and Cios, 2002a; Kurgan and Cios, 2003a). These papers propose and provide detailed description of the DataSqueezer algorithm. The benchmarking results of the DataSqueezer algorithms are based on (Kurgan and Cios, 2003a). Also, the results which were used in comparison of the DataSqueezer algorithm with other inductive ML algorithms were taken from (Cios and Kurgan, 2002a). The description of the CAIM algorithm is based on (Kurgan and Cios, 2001; Kurgan and Cios, 2002b; Kurgan and Cios, 2003b). The benchmarking results of the CAIM were taken from (Kurgan and Cios, 2002b).

The fourth chapter provides detailed description of the MetaSqueezer system. The description of the system together with its benchmarking tests were based on (Kurgan and Cios, 2003a). Also, two papers were used to provide results that were used in comparison of the MetaSqueezer system with other inductive ML algorithms (Kurgan and Cios, 2002a; Cios and Kurgan, 2003a).

The fifth chapter provides description of the project where the MetaSqueezer system is used to analyze data concerning CF patients. The project was carried using the six step DMKD process model. Experience from application of the model to another medical problem, concerning analysis of cardiac SPECT data, was used to proceed with and describe the project (Kurgan et al., 2001). Also, description of the process model was taken from (Cios and Kurgan, 2002b). Implementation and description of the F-CAIM algorithm, which was used to discretize the CF data, was based on (Kurgan and Cios, 2003b). Finally, another paper was used to design the attribute and selector ranking tables that were used to display and analyze results generated in the project (Cios and Kurgan, 2002a).

List of Relevant Publications:

Cios, K. J. and Kurgan, L., Hybrid Inductive Machine Learning: An Overview of CLIP Algorithms. In: Jain, L.C. & Kacprzyk, J. (Eds.), *New Learning Paradigms in Soft Computing*, Physica-Verlag (Springer), pp. 276-322, 2001

Cios, K.J., and Kurgan, L., Hybrid Inductive Machine Learning Algorithm that Generates Inequality Rules, *Information Sciences*, Special Issue on Soft Computing Data Mining, accepted, 2002a

Cios, K. J., and Kurgan, L., Trends in Data Mining and Knowledge Discovery, In: Pal N.R., Jain, L.C. and Teoderesku, N. (Eds.), *Knowledge Discovery in Advanced Information Systems*, Springer, to appear, 2002b

Kurgan, L.A., Cios, K.J., Tadeusiewicz, R., Ogiela, M. and Goodenday, L.S., Knowledge Discovery Approach to Automated Cardiac SPECT Diagnosis, *Artificial Intelligence in Medicine*, 23(2), pp. 149-169, 2001

Kurgan, L., and Cios, K.J., Discretization Algorithm that Uses Class-Attribute Interdependence Maximization, *Proceedings of the 2001 International Conference on Artificial Intelligence* (IC-AI 2001), Las Vegas, Nevada, pp.980-987, 2001

Kurgan, L., and Cios, K.J., DataSqueezer Algorithm that Generates Small Number of Short Rules, *IEE Proceedings: Vision, Image and Signal Processing*, submitted, 2002a

Kurgan, L., and Cios, K.J., CAIM Discretization Algorithm, *IEEE Transactions of Knowledge and Data Engineering*, accepted, 2002b

Kurgan, L., & Cios, K. J., Meta-Mining Architecture for Learning from Supervised Data, submitted, 2003a

Kurgan, L., and Cios, K.J., Fast Class-Attribute Interdependence Maximization (CAIM) Discretization Algorithm, *Proceeding of the 2003 International Conference on Machine Learning and Applications* (ICMLA'03), pp.30-36, Los Angeles, 2003b

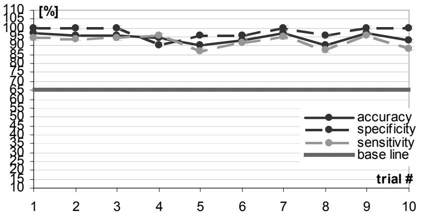
Appendix C

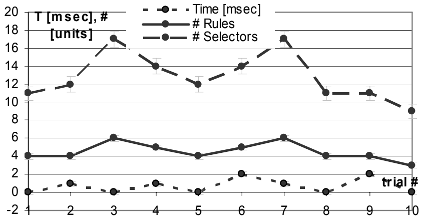
Detailed Test Results

DATA SET NAME: **bcw**

- results of tests with the **DataSqueezer** algorithm

Trials	1	2	3	4	5	6	7	8	9	10	stdev	MEAN
Accuracy	97.1	95.7	95.7	94.3	90	92.9	97.1	90	97.1	92.8	*2.78*	**94.3**
Specificity	100	100	100	90.5	95.7	95.5	100	95.5	100	100	*3.3*	**97.7**
Sensitivity	94.3	93.8	94.2	95.9	87.2	91.7	94.9	87.5	95.9	88.4	*3.45*	**92.4**
Time [msec]	0	1	0	1	0	2	1	0	2	0	*0.82*	**0.7 msec**
# Rules	4	4	6	5	4	5	6	4	4	3	*0.97*	**4.5**
# Selectors	11	12	17	14	12	14	17	11	11	9	*2.66*	**12.8**

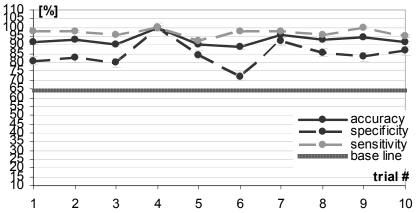


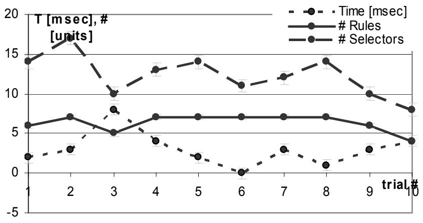


DATA SET NAME: **bcw**

- results of tests with the **MetaSqueezer** system

Trials	1	2	3	4	5	6	7	8	9	10	stdev	MEAN
Accuracy	91.4	92.9	90	100	90	88.6	95.7	92.9	94.3	91.3	*3.34*	**92.7**
Specificity	80.8	82.6	80	100	84.2	72	92	85.7	83.3	86.7	*7.44*	**84.7**
Sensitivity	97.7	97.9	95.6	100	92.2	97.8	97.8	95.9	100	94.9	*2.4*	**97.0**
Time [msec]	2	3	8	4	2	0	3	1	3	4	*2.16*	**3 msec**
# Rules	6	7	5	7	7	7	7	7	6	4	*1.06*	**6.3**
# Selectors	14	17	10	13	14	11	12	14	10	8	*2.63*	**12.3**

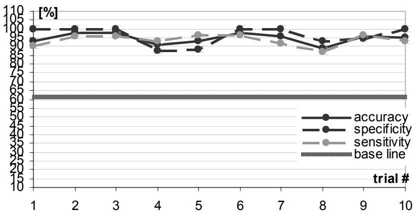


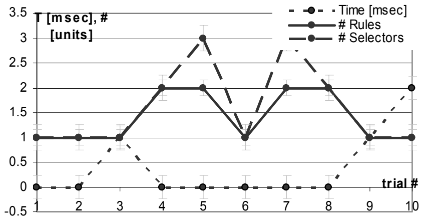


DATA SET NAME: **vot**

- results of tests with the **DataSqueezer** algorithm

Trials	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*stdev*	*MEAN*
Accuracy	93.2	97.7	97.7	90.9	93.2	97.7	95.5	88.6	95.5	94.7	*3.05*	**94.5**
Specificity	100	100	100	87.5	88.2	100	100	92.9	94.4	100	*5.16*	**96.3**
Sensitivity	90	95.5	95.8	92.9	96.3	96.4	91.7	86.7	96.2	92.9	*3.26*	**93.4**
Time [msec]	0	0	1	0	0	0	0	0	1	2	0.7	**0.4 msec**
# Rules	1	1	1	2	2	1	2	2	1	1	*0.52*	**1.4**
# Selectors	1	1	1	2	3	1	3	2	1	1	*0.84*	**1.6**

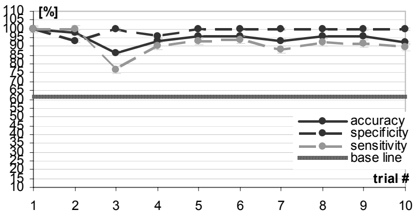


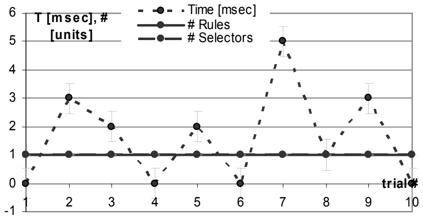


DATA SET NAME: **vot**

- results of tests with the **MetaSqueezer** system

Trials	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*stdev*	*MEAN*
Accuracy	100	97.7	86.4	93.2	95.5	95.5	93.2	95.5	95.5	92.1	*3.65*	**94.4**
Specificity	100	93.3	100	95.7	100	100	100	100	100	100	*2.39*	**98.9**
Sensitivity	100	100	76.9	90.5	92.9	93.8	88	92.3	91.7	89.3	*6.52*	**91.5**
Time [msec]	0	3	2	0	2	0	5	1	3	0	1.71	**1.6 msec**
# Rules	1	1	1	1	1	1	1	1	1	1	*0*	**1**
# Selectors	1	1	1	1	1	1	1	1	1	1	*0*	**1**

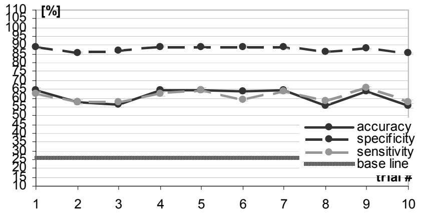


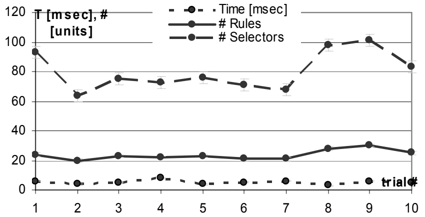


DATA SET NAME: **veh**

- results of tests with the **DataSqueezer** algorithm

Trials	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	stdev	*MEAN*
Accuracy	64.7	57.6	56.5	64.7	64.7	63.5	64.7	55.3	63.5	55.6	*4.24*	**61.1**
Specificity	89	85.8	87	89.2	88.9	88.9	88.9	86	88	85.3	*1.56*	**87.7**
Sensitivity	62.7	57.9	57.3	62.2	64.5	59.1	63.7	58.1	65.7	57.7	*3.19*	**60.9**
Time [msec]	6	4	5	8	4	5	6	3	6	5	1.4	**5.2 msec**
# Rules	24	20	23	22	23	21	21	28	30	25	*3.2*	**23.7**
# Selectors	93	64	75	73	76	71	68	98	101	83	*13*	**80.2**

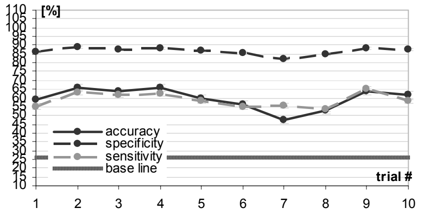


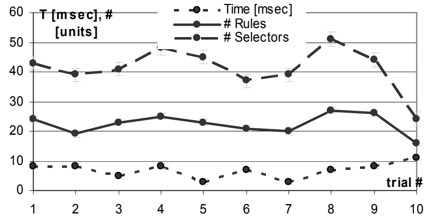


DATA SET NAME: **veh**

- results of tests with the **MetaSqueezer** system

Trials	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*stdev*	*MEAN*
Accuracy	58.8	65.9	63.5	65.9	60	56.5	47.1	52.9	63.5	61.7	*6.02*	**59.6**
Specificity	86.3	88.8	87.6	88.2	86.9	85.3	82	84.6	88	87.3	*2.04*	**86.5**
Sensitivity	54.8	62.9	61.7	62.2	58	55.1	55.4	53.7	65.2	58	*4.02*	**58.7**
Time [msec]	8	8	5	8	3	7	3	7	8	11	2.49	**6.8 msec**
# Rules	24	19	23	25	23	21	20	27	26	16	*3.41*	**22.4**
# Selectors	43	39	41	48	45	37	39	51	44	24	*7.39*	**41.1**

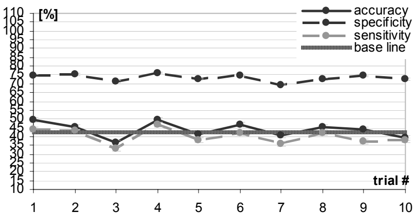


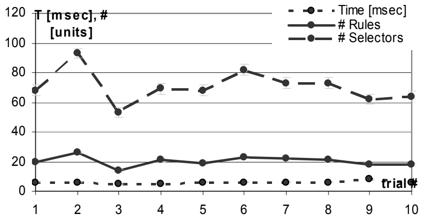


DATA SET NAME: **cmc**

- results of tests with the **DataSqueezer** algorithm

Trials	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	stdev	*MEAN*
Accuracy	49.3	45.3	36.5	49.3	41.2	46.6	40.5	45.3	43.9	39	*4.31*	**43.7**
Specificity	74.5	75.1	71.1	76.3	72.9	74.5	69.5	72.5	74.3	72.3	*2.03*	**73.3**
Sensitivity	44.2	43	33.2	46.9	38	42	35.8	42	37.5	37.8	*4.24*	**40.0**
Time [msec]	6	6	5	5	6	6	6	6	8	6	0.82	**6 msec**
# Rules	20	26	14	21	19	23	22	21	18	18	*3.26*	**20.2**
# Selectors	68	93	53	69	68	82	73	73	62	64	*11*	**70.5**

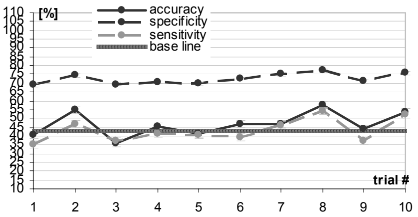


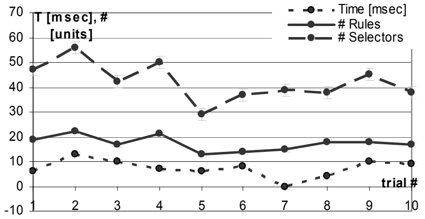


DATA SET NAME: **cmc**

- results of tests with the **MetaSqueezer** system

Trials	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*stdev*	MEAN
Accuracy	40.5	54.7	35.8	45.3	41.2	46.6	46.6	57.4	43.9	53.2	*6.81*	**46.5**
Specificity	68.9	74.5	69.3	70.5	69.7	72.6	75.2	77.1	70.9	75.8	*2.99*	**72.4**
Sensitivity	35.3	46.4	37	41	40.3	39.5	45.8	54	37	51.9	*6.44*	**42.8**
Time [msec]	6	13	10	7	6	8	0	4	10	9	3.62	**7.3 msec**
# Rules	19	22	17	21	13	14	15	18	18	17	*2.88*	**17.4**
# Selectors	47	56	42	50	29	37	39	38	45	38	*7.67*	**42.1**

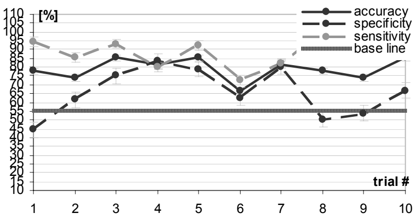


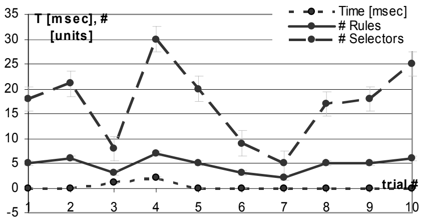


DATA SET NAME: **hea**

- results of tests with the **DataSqueezer** algorithm

Trials	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*stdev*	*MEAN*
Accuracy	77.8	74.1	85.2	81.5	85.2	66.7	81.5	77.8	74.1	85.2	*6.06*	**78.9**
Specificity	44.4	61.5	75	83.3	78.6	62.5	80	50	53.8	66.7	*13.5*	**65.6**
Sensitivity	94.4	85.7	93.3	80	92.3	72.7	82.4	100	92.9	94.4	*8.34*	**88.8**
Time [msec]	0	0	1	2	0	0	0	0	0	0	0.67	**0.3 msec**
# Rules	5	6	3	7	5	3	2	5	5	6	*1.57*	**4.7**
# Selectors	18	21	8	30	20	9	5	17	18	25	*7.81*	**17.1**

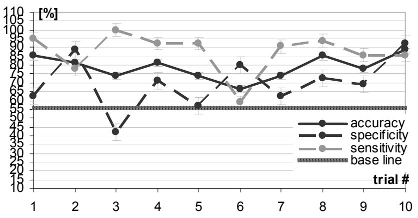


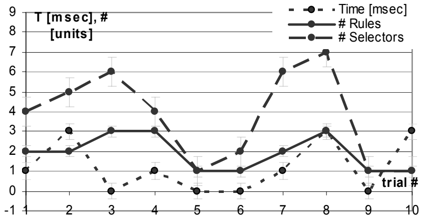


DATA SET NAME: **hea**

- results of tests with the **MetaSqueezer** system

Trials	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*stdev*	*MEAN*
Accuracy	85.2	81.5	74.1	81.5	74.1	66.7	74.1	85.2	77.8	88.9	*6.77*	**78.9**
Specificity	62.5	88.9	41.7	71.4	57.1	80	62.5	72.7	69.2	92.3	*15.1*	**69.8**
Sensitivity	94.7	77.8	100	92.3	92.3	58.8	90.9	93.8	85.7	85.7	*11.7*	**87.2**
Time [msec]	1	3	0	1	0	0	1	3	0	3	1.32	**1.2 msec**
# Rules	2	2	3	3	1	1	2	3	1	1	*0.88*	**1.9**
# Selectors	4	5	6	4	1	2	6	7	1	1	*2.31*	**3.7**

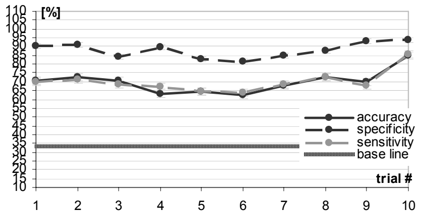


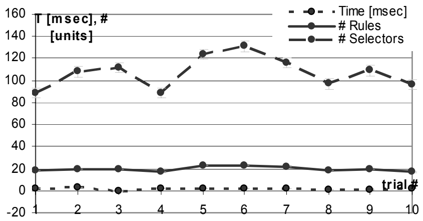


DATA SET NAME: **bos**

- results of tests with the **DataSqueezer** algorithm

Trials	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*stdev*	*MEAN*
Accuracy	70.6	72.5	70.6	63.3	64.7	62.7	68	72.5	70	85.1	*6.43*	**70.0**
Specificity	90.2	90.9	84.1	89.8	82.5	81.5	85.1	87.6	93.3	93.4	*4.33*	**87.8**
Sensitivity	69.7	71.3	68.6	67.4	64.5	63.7	68.4	72.5	67.8	85.6	*6.13*	**69.9**
Time [msec]	2	3	0	2	2	2	2	1	1	2	0.82	**1.7 msec**
# Rules	18	20	20	17	23	23	22	18	20	17	*2.3*	**19.8**
# Selectors	89	108	112	89	124	131	116	97	109	96	*14.3*	**107**

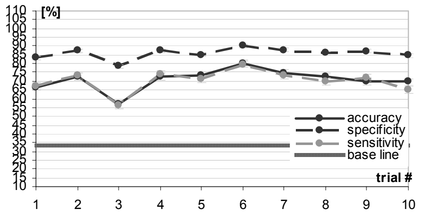


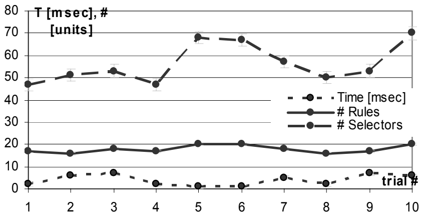


DATA SET NAME: **bos**

- results of tests with the **MetaSqueezer** system

Trials	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*stdev*	*MEAN*
Accuracy	66.7	72.5	56.9	72.5	73.5	80.4	74.5	72.5	70	69.6	*6.11*	**70.9**
Specificity	83.3	87.3	78.4	87.3	84.9	90.2	87.6	85.9	86.9	84.5	*3.2*	**85.6**
Sensitivity	67	73.4	56.3	74.1	71.1	79.7	73.3	70	71.9	64.9	*6.34*	**70.2**
Time [msec]	2	6	7	2	1	1	5	2	7	6	2.51	**3.9 msec**
# Rules	17	16	18	17	20	20	18	16	17	20	*1.6*	**17.9**
# Selectors	47	51	53	47	68	67	57	50	53	70	*8.83*	**56.3**

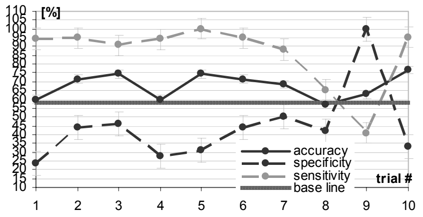


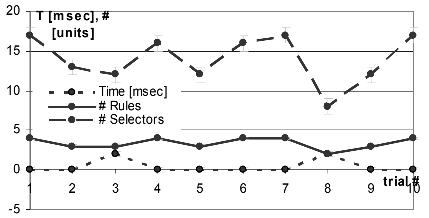


DATA SET NAME: **bld**

- results of tests with the **DataSqueezer** algorithm

Trials	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*stdev*	*MEAN*
Accuracy	60	71.4	74.3	60	74.3	71.4	68.6	57.1	62.9	76.7	*7.06*	**67.7**
Specificity	23.5	43.8	46.2	27.8	30.8	43.8	50	41.7	100	33.3	*21.5*	**44.1**
Sensitivity	94.4	94.7	90.9	94.1	100	94.7	88.2	65.2	40.9	95.2	*18.5*	**85.9**
Time [msec]	0	0	2	0	0	0	0	2	0	0	0.84	**0.4 msec**
# Rules	4	3	3	4	3	4	4	2	3	4	*0.7*	**3.4**
# Selectors	17	13	12	16	12	16	17	8	12	17	*3.06*	**14.0**

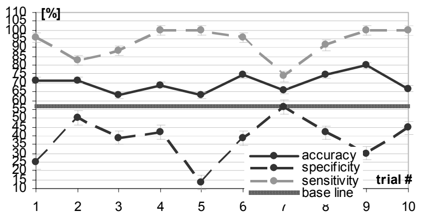


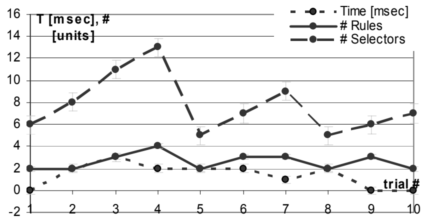


DATA SET NAME: **bld**

- results of tests with the **MetaSqueezer** system

Trials	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*stdev*	*MEAN*
Accuracy	71.4	71.4	62.9	68.6	62.9	74.3	65.7	74.3	80	66.7	*5.52*	**69.8**
Specificity	25	50	38.9	42.1	13.3	38.5	56.3	41.7	30	44.4	*12.4*	**38.0**
Sensitivity	95.7	82.6	88.2	100	100	95.5	73.7	91.3	100	100	*8.91*	**92.7**
Time [msec]	0	2	3	2	2	2	1	2	0	0	1.07	**1.4 msec**
# Rules	2	2	3	4	2	3	3	2	3	2	*0.7*	**2.6**
# Selectors	6	8	11	13	5	7	9	5	6	7	*2.63*	**7.7**

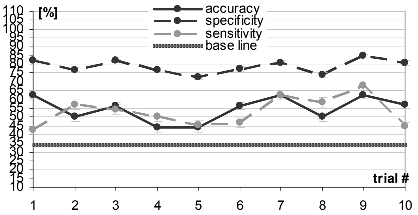


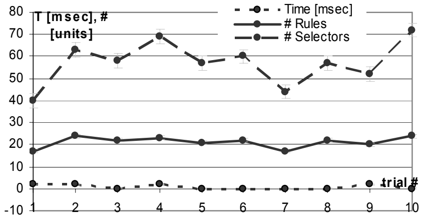


DATA SET NAME: **tae**

- results of tests with the **DataSqueezer** algorithm

Trials	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*stdev*	*MEAN*
Accuracy	62.5	50	56.3	43.8	43.8	56.3	62.5	50	62.5	57.1	*7.28*	**54.5**
Specificity	81.9	77	81.8	77	72.7	77.5	81	74	84.6	80.6	*3.77*	**78.8**
Sensitivity	42.6	57.2	54.2	49.8	45.6	46.8	62.4	58.3	67.5	44.4	*8.38*	**52.9**
Time [msec]	2	2	0	2	0	0	0	0	2	0	1.03	**0.8 msec**
# Rules	17	24	22	23	21	22	17	22	20	24	*2.53*	**21.2**
# Selectors	40	63	58	69	57	60	44	57	52	72	*9.99*	**57.2**

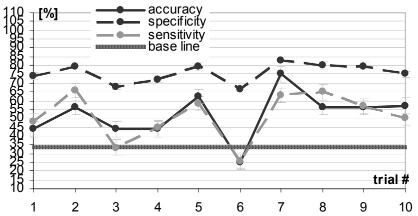


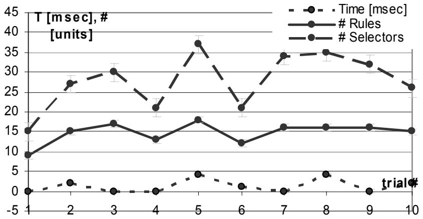


DATA SET NAME: **tae**

- results of tests with the **MetaSqueezer** system

Trials	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*stdev*	*MEAN*
Accuracy	43.8	56.3	43.8	43.8	62.5	25	75	56.3	56.3	57.1	*13.6*	**52.0**
Specificity	73.9	79.5	68.1	72.1	79.4	66.4	82.9	80.1	79.5	75	*5.54*	**75.7**
Sensitivity	48.4	65.8	33.3	44.4	58.3	25.6	63	64.8	56.9	50	*13.5*	**51.1**
Time [msec]	0	2	0	0	4	1	0	4	0	2	1.64	**1.3 msec**
# Rules	9	15	17	13	18	12	16	16	16	15	*2.67*	**14.7**
# Selectors	15	27	30	21	37	21	34	35	32	26	*7.13*	**27.8**

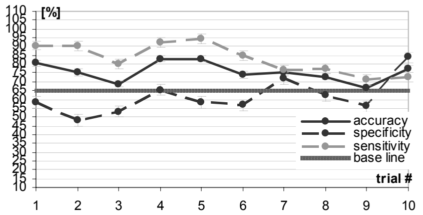


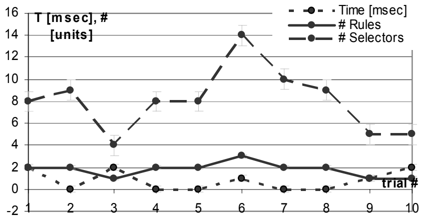


DATA SET NAME: **pid**

- results of tests with the **DataSqueezer** algorithm

Trials	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*stdev*	*MEAN*
Accuracy	80.5	75.3	68.8	83.1	83.1	74	75.3	72.7	66.2	77.3	*5.62*	**75.7**
Specificity	58.3	48.1	53.1	65.4	58.3	56.7	72	62.5	56	83.9	*10.3*	**61.4**
Sensitivity	90.6	90	80	92.2	94.3	85.1	76.9	77.4	71.2	72.7	*8.48*	**83.0**
Time [msec]	2	0	2	0	0	1	0	0	1	2	0.92	**0.8 msec**
# Rules	2	2	1	2	2	3	2	2	1	1	*0.63*	**1.8**
# Selectors	8	9	4	8	8	14	10	9	5	5	*2.91*	**8.0**

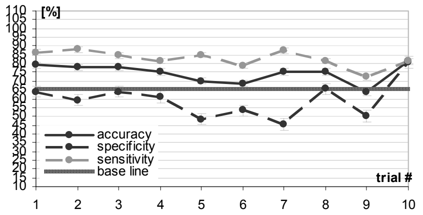


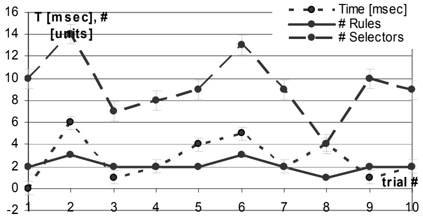


DATA SET NAME: **pid**

- results of tests with the **MetaSqueezer** system

Trials	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*stdev*	*MEAN*
Accuracy	79.2	77.9	77.9	75.3	70.1	68.8	75.3	75.3	63.6	81.3	*5.41*	**74.5**
Specificity	64	59.3	64	60.9	48.4	53.3	45.5	65.5	50	80.8	*10.4*	**59.2**
Sensitivity	86.5	88	84.6	81.5	84.8	78.7	87.3	81.3	72.3	81.6	*4.71*	**82.7**
Time [msec]	0	6	1	2	4	5	2	4	1	2	1.95	**2.7 msec**
# Rules	2	3	2	2	2	3	2	1	2	2	*0.57*	**2.1**
# Selectors	10	14	7	8	9	13	9	4	10	9	*2.83*	**9.3**

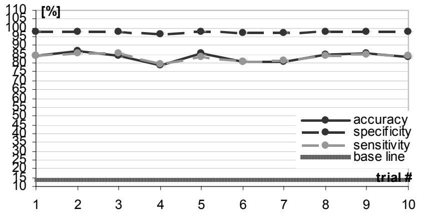


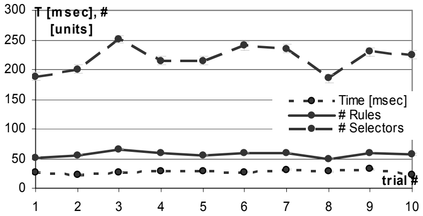


DATA SET NAME: **seg**

- results of tests with the **DataSqueezer** algorithm

Trials	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*stdev*	*MEAN*
Accuracy	84	86.6	84.4	78.8	85.7	81	81	84.8	85.3	83.5	*2.49*	**83.5**
Specificity	97.6	97.9	97.8	96.7	98	97.3	97.2	97.5	97.6	97.7	*0.39*	**97.5**
Sensitivity	84.2	85.4	85.2	79.2	83.7	80.9	81.1	84.3	84.5	83.9	*2.09*	**83.2**
Time [msec]	27	23	27	28	28	27	31	28	32	22	3.06	**27.3 msec**
# Rules	51	55	66	59	56	60	60	50	59	57	*4.67*	**57.3**
# Selectors	188	201	251	215	215	240	235	185	230	225	*22*	**219**

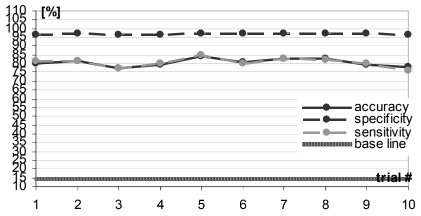


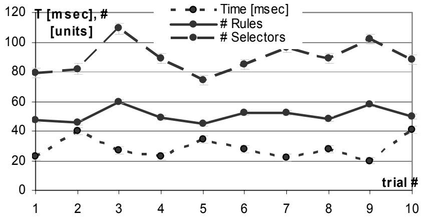


DATA SET NAME: **seg**

- results of tests with the **MetaSqueezer** system

Trials	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*stdev*	*MEAN*
Accuracy	80.1	81.4	77.1	79.7	84	80.5	82.7	82.7	79.7	77.9	*2.17*	**80.6**
Specificity	96.6	96.9	96.2	96.6	97.4	96.8	97.1	97.1	96.8	96.3	*0.38*	**96.8**
Sensitivity	81.5	81.4	77.6	79.8	84.5	80.2	82.5	81.9	79.9	75.8	*2.48*	**80.5**
Time [msec]	23	40	27	23	34	28	22	28	20	41	7.43	**28.6 msec**
# Rules	47	46	60	49	45	52	52	48	58	50	*4.97*	**50.7**
# Selectors	79	82	109	89	74	85	96	89	102	88	*10.6*	**89.3**





DATA SET NAME: **dna**

results of tests with the **DataSqueezer** algorithm

Trials	*1*
Accuracy	**92.0**
Sensitivity	**92.4**
Specificity	**97.1**
Base Line	**51.2**
Time	**56 msec**
# Rules	**39**
# Selectors	**97**

results of tests with the **MetaSqueezer** system

Trials	*1*
Accuracy	**89.6**
Sensitivity	**89.3**
Specificity	**94.9**
Base Line	**51.2**
Time	**1 sec 2 msec**
# Rules	**34**
# Selectors	**53**

DATA SET NAME: **led**

results of tests with the **DataSqueezer** algorithm

Trials	*1*
Accuracy	**68.4**
Sensitivity	**68.2**
Specificity	**96.5**
Base Line	**10.8**
Time	**17 msec**
# Rules	**51**
# Selectors	**194**

results of tests with the **MetaSqueezer** system

Trials	*1*
Accuracy	**68.9**
Sensitivity	**68.9**
Specificity	**96.6**
Base Line	**10.8**
Time	**22 msec**
# Rules	**51**
# Selectors	**141**

DATA SET NAME: **sat**

results of tests with the **DataSqueezer** algorithm

Trials	*1*
Accuracy	**79.5**
Sensitivity	**77.7**
Specificity	**96.2**
Base Line	**24.2**
Time	**95 msec**
# Rules	**57**
# Selectors	**257**

results of tests with the **MetaSqueezer** system

Trials	*1*
Accuracy	**74.4**
Sensitivity	**73.4**
Specificity	**94.5**
Base Line	**24.2**
Time	**1 sec 7 msec**
# Rules	**55**
# Selectors	**104**

DATA SET NAME: **smo**

results of tests with the **DataSqueezer** algorithm

Trials	*1*
Accuracy	**68.3**
Sensitivity	**33.3**
Specificity	**66.7**
Base Line	**70.2**
Time	**5 msec**
# Rules	**6**
# Selectors	**12**

results of tests with the **MetaSqueezer** system

Trials	*1*
Accuracy	**66.8**
Sensitivity	**32.6**
Specificity	**68.7**
Base Line	**70.2**
Time	**6 msec**
# Rules	**3**
# Selectors	**11**

DATA SET NAME: **thy**

results of tests with the **DataSqueezer** algorithm

Trials	1
Accuracy	**96.2**
Sensitivity	**94.6**
Specificity	**98.8**
Base Line	**32.5**
Time	**6 msec**
# Rules	**7**
# Selectors	**28**

results of tests with the **MetaSqueezer** system

Trials	1
Accuracy	**96.2**
Sensitivity	**85.9**
Specificity	**98.7**
Base Line	**32.5**
Time	**7 msec**
# Rules	**6**
# Selectors	**6**

DATA SET NAME: **wav**

results of tests with the **DataSqueezer** algorithm

Trials	1
Accuracy	**76.7**
Sensitivity	**76.6**
Specificity	**88.8**
Base Line	**35.8**
Time	**2 msec**
# Rules	**22**
# Selectors	**65**

results of tests with the **MetaSqueezer** system

Trials	1
Accuracy	**77.6**
Sensitivity	**77.5**
Specificity	**89.0**
Base Line	**35.8**
Time	**2 msec**
# Rules	**16**
# Selectors	**16**

DATA SET NAME: **mush**

results of tests with the **DataSqueezer** algorithm

Trials	1
Accuracy	**99.9**
Sensitivity	**99.7**
Specificity	**100**
Base Line	**53.3**
Time	**6 msec**
# Rules	**8**
# Selectors	**21**

results of tests with the **MetaSqueezer** system

Trials	1
Accuracy	**99.5**
Sensitivity	**99**
Specificity	**100**
Base Line	**53.3**
Time	**19 msec**
# Rules	**6**
# Selectors	**16**

DATA SET NAME: **adult**

results of tests with the **DataSqueezer** algorithm

Trials	1
Accuracy	**81.5**
Sensitivity	**93.9**
Specificity	**41.4**
Base Line	**75.9**
Time	**17 sec 83 msec**
# Rules	**61**
# Selectors	**395**

results of tests with the **MetaSqueezer** system

Trials	1
Accuracy	**80.6**
Sensitivity	**94.6**
Specificity	**33.4**
Base Line	**75.9**
Time	**10 sec 35 msec**
# Rules	**19**
# Selectors	**64**

DATA SET NAME: **forc**

results of tests with the **DataSqueezer** algorithm

Trials	*1*
Accuracy	**54.8**
Sensitivity	**56.2**
Specificity	**90.0**
Base Line	**14.3**
Time	**25 sec 98 msec**
# Rules	**59**
# Selectors	**2105**

results of tests with the **MetaSqueezer** system

Trials	*1*
Accuracy	**55.3**
Sensitivity	**35.8**
Specificity	**88.8**
Base Line	**14.3**
Time	**18 sec 57 msec**
# Rules	**33**
# Selectors	**699**

DATA SET NAME: **cid**

- results of tests with the **DataSqueezer** algorithm

Trials	*1*
Accuracy	**90.5**
Sensitivity	**93.5**
Specificity	**45.4**
Base Line	**93.8**
Time	**4 min 25 sec 09 msec**
# Rules	**15**
# Selectors	**95**

Train data size	**1000**	**2000**	**4000**	**8000**	**16000**	**32000**	**64000**	**128000**	199523
Train data size - Ratio	**---**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**1.56**
Time	**30msec**	**58msec**	**98msec**	**2sec 47msec**	**6sec 90msec**	**18sec 89msec**	**47sec 13msec**	**2min 16sec 29msec**	**4min 25sec 09msec**
Time - Ratio	**---**	**1.93**	**1.69**	**2.52**	**2.79**	**2.74**	**2.50**	**2.95**	**1.94**
Accuracy	**87.9**	**89.9**	**88.1**	**87.7**	**87.9**	**89.3**	**89.4**	**89.1**	**90.5**
Sensitivity	**90.3**	**92.3**	**90.7**	**90.3**	**90.5**	**92.1**	**92.2**	**91.8**	**93.5**
Specificity	**52.0**	**45.6**	**48.7**	**49.1**	**48.4**	**46.5**	**46.4**	**47.3**	**45.4**
# Rules	**11**	**13**	**10**	**12**	**12**	**13**	**14**	**13**	**15**
# Selectors	**74**	**80**	**57**	**77**	**74**	**81**	**88**	**83**	**95**

- results of tests with the **MetaSqueezer** system

Trials	*1*
Accuracy	**90.2**
Sensitivity	**93.0**
Specificity	**49.0**
Base Line	**93.8**
Time	**4 min 33 sec 01 msec**
# Rules	**6**
# Selectors	**34**

Train data size	**1000**	**2000**	**4000**	**8000**	**16000**	**32000**	**64000**	**128000**	199523
Train data size - Ratio	**---**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**1.56**
Time	**65msec**	**1sec 27msec**	**2sec 31msec**	**4sec 72msec**	**10sec 44msec**	**24sec 71msec**	**55sec 43msec**	**2min 31sec 71msec**	**4min 33sec 01msec**
Time - Ratio	**---**	**1.95**	**1.81**	**2.04**	**2.21**	**2.34**	**2.24**	**2.74**	**1.80**
Accuracy	**93.6**	**90.6**	**90.3**	**92.0**	**92.5**	**91.7**	**90.5**	**90.9**	**90.2**
Sensitivity	**99.6**	**94.9**	**93.8**	**96.3**	**97.2**	**95.9**	**93.7**	**94.1**	**93.0**
Specificity	**1.9**	**25.7**	**37.1**	**26.5**	**20.6**	**28.1**	**41.8**	**41.7**	**49.0**
# Rules	**38**	**11**	**14**	**17**	**18**	**13**	**11**	**11**	**6**
# Selectors	**173**	**67**	**76**	**106**	**98**	**66**	**72**	**67**	**34**

Appendix D

Results of Mining CF data using DataSqueezer Algorithm

Results for task 1

The following attribute and selector importance table was generated by the DataSqueezer algorithm, when applying to the input data for task 1 and using the same parameters as the parameters as for the MetaSqueezer system. Evaluation of findings, which was performed by Dr. Frank Accurso and Marci Sontag, is shown in the marks column.

ATRIBUTE	VALUE	MARK	CLASS IMPROV	CLASS NOCHANGE	CLASS SLOWDEGRAD	CLASS FASTDEGRAD
CFtypes (cf)			▨	▨	▨	▨
CFtypes (cf)	Type1		▨	▨	▨	▨
CFtypes (cf)	Type2		▨		▨	▨
TemporalIntervals (cf)	4				▨	
TemporalIntervals (cf)	5				▨	▨
vis_hgt1 (vis)			▨	▨	▨	▨
vis_hgt1 (vis)	[105.00,180.00)		■	■	■	■
vis_wght1 (vis)					▨	
vis_wght1 (vis)	[22.10,54.40)		▨	▨	▨	▨
dob (dem)				▨	▨	
dob (dem)	82till92		▨	▨	▨	
dob (dem)	before82				▨	
race (dem)			▨	▨	▨	▨
race (dem)	Caucasian		■	■	■	■
race (dem)	Black	**3+**	▨			
group (dem)			▨	▨	▨	
group (dem)	C				▨	
group (dem)	NBS	**3+**	▨	▨	▨	
group (dem)	MI	**3+**	▨			
marital (dem)			▨	▨	▨	▨
marital (dem)	Married		■	▨	▨	▨
motage (dem)		**3+**	▨	▨		
motage (dem)	[22.50,48.50)		▨	▨	▨	
birthord (dem)			▨	▨	▨	▨
birthord (dem)	3		▨			▨
birthord (dem)	2		▨	▨	▨	▨
birthord (dem)	1		■	▨	▨	▨
birthord (dem)	4				▨	
numsib (dem)			▨	▨	▨	▨
numsib (dem)	2		▨		▨	▨
numsib (dem)	1		▨	▨	▨	▨
numsib (dem)	3				▨	
numsib (dem)	0		▨	▨		
numsib (dem)	4				▨	▨

cfsib (dem)			▒	▒		▒
cfsib (dem)	0		▒	▒	▒	▒
deltype (dem)			▒	▒	▒	▒
deltype (dem)	vagnl		▒	▒	▒	▒
deltype (dem)	csemg		▒			▒
deltype (dem)	cspln					▒
deltype (dem)	unk			▒		
mecil (dem)	no		▒	▒	▒	
irt (dia)	No				▒	▒
irt (dia)	Yes		▒	▒	▒	
FalseNeg (dia)					▒	
FalseNeg (dia)	-				▒	
FalseNeg (dia)	No		▒	▒	▒	
Clinic (dia)			▒	▒	▒	▒
Clinic (dia)	Billings		▒	▒	▒	
Clinic (dia)	Adult		▒		▒	▒
Clinic (dia)	Denver		■	■	■	■
Clinic (dia)	Colo.Spgs		▒	▒	▒	▒
Clinic (dia)	GreatFalls				▒	▒
irt1 (dia)	[389.00,766.00)		▒			
irt1 (dia)	[79.50,373.00)		▒	▒	▒	▒
age@irt1 (dia)			▒	▒		
age@irt1 (dia)	[-0.50,5.50)		▒	▒		▒
age@irt1 (dia)	[5.50,8.50)		▒			
irt2 (dia)	[87.50,322.00)		▒	▒		▒
swetna1 (dia)			▒	▒	▒	▒
swetna1 (dia)	[54.50,152.00)		■	■	■	■
swetk1 (dia)					▒	▒
swetk1 (dia)	[24.50,46.00)			▒	▒	
swetk1 (dia)	[4.50,16.50)			▒		
swetk1 (dia)	[16.50,24.50)		▒		▒	▒
swetcl1 (dia)			▒	▒	▒	▒
swetcl1 (dia)	[101.00,157.00)		▒	▒	▒	▒
swetcl1 (dia)	[-11.00,95.50)	**3+**	▒			
swetna2 (dia)				▒	▒	
swetna2 (dia)	[22.00,101.00)			■	▒	
swetk2 (dia)	[19.50,58.50)			▒	▒	
swetcl2 (dia)				▒		
swetcl2 (dia)	[26.50,108.00)			▒	▒	
swetk3 (dia)	[5.50,26.50)				▒	
swetk4 (dia)	[10.50,39.50)				▒	
SOURCE (cul)	Sputum				▒	▒
SOURCE (cul)	ThroatCulture		▒		▒	▒
tobraresistent? (cul)					▒	
tobraresistent? (cul)	No		▒	▒	▒	▒
ciproresistant? (cul)					▒	
ciproresistant? (cul)	No		▒	▒	▒	▒
meropenemresistant? (cul)					▒	
meropenemresistant? (cul)	No		▒	▒	▒	▒
WAZ (per)			▒	▒	▒	▒
WAZ (per)	[-1.93,1.51)		■	■	■	■
HAZ (per)			▒	▒	▒	▒
HAZ (per)	[-1.87,2.50)		■	■	■	■
BASEpower (cul)	[-1.00,2E4)		▒		▒	▒
BASEpower2 (cul)	[-1.75,7.5E6)		▒		▒	▒
BASEpower3 (cul)					▒	
BASEpower3 (cul)	[-1.75,1.1E8)		▒		▒	▒
BASEpower4 (cul)					▒	
BASEpower4 (cul)	[-5E3,1.25E5)		▒		▒	▒
age@diagnosis (dia)			▒	▒	▒	▒

age@diagnosis (dia)	025till2				▨	
age@diagnosis (dia)	below025		▨	■	▨	▨

Results for task 2

The following attribute and selector importance table was generated by the DataSqueezer algorithm, when applying to the input data for task 2 and using the same parameters as the parameters as for the MetaSqueezer system. Evaluation of findings, which was performed by Dr. Frank Accurso and Marci Sontag, is shown in the marks column.

ATRIBUTE	VALUE	MARK	CLASS TYPE1	CLASS TYPE2	CLASS TYPE4
CFpace (cf)					▨
CFpace (cf)	NoChange		▨	▨	▨
CFpace (cf)	SlowDegrad				▨
CFpace (cf)	Improv	**3+**			▨
TemporalIntervals (cf)	4				▨
vis_hgt1 (vis)			▨	▨	▨
vis_hgt1 (vis)	[102.00,188.00)		■	■	■
vis_wght1 (vis)			▨	▨	▨
vis_wght1 (vis)	[15.30,53.30)		■	■	▨
PulseOximetry (vis)	[92.50,101.00)			▨	▨
dob (dem)	82till92			▨	
race (dem)			▨	▨	▨
race (dem)	Caucasian		■	■	■
race (dem)	Unknown				▨
group (dem)			▨	▨	▨
group (dem)	C	**2+**			▨
group (dem)	NBS	**2+**	▨	▨	▨
marital (dem)			▨	▨	▨
marital (dem)	Married		▨	▨	■
marital (dem)	Unknown			▨	
motage (dem)					▨
motage (dem)	[25.50,37.50)		▨	▨	▨
motage (dem)	[15.10,24.50)			▨	
birthord (dem)			▨	▨	▨
birthord (dem)	3		▨	▨	▨
birthord (dem)	2		▨	▨	▨
birthord (dem)	1		▨	▨	▨
birthord (dem)	4			▨	
numsib (dem)			▨	▨	▨
numsib (dem)	2		▨	▨	▨
numsib (dem)	1		▨	▨	▨
numsib (dem)	0		▨	▨	▨
cfsib (dem)			▨	▨	▨
cfsib (dem)	1			▨	▨
cfsib (dem)	0		■	▨	▨
deltype (dem)			▨	▨	▨
deltype (dem)	vagnl		▨	▨	▨
deltype (dem)	unk			▨	
mecil (dem)			▨	▨	

mecil (dem)	no		▒	▒	▒
dcmot (dem)			▒	▒	▒
dcmot (dem)	yes		▒	▒	▒
Clinic (dia)			▒	▒	▒
Clinic (dia)	Billings		▒		▒
Clinic (dia)	Adult		▒		
Clinic (dia)	Denver		█	█	█
Clinic (dia)	Colo.Spgs		▒	▒	
Clinic (dia)	GreatFalls				▒
irt1 (dia)	[112.00,285.00)		▒	▒	▒
age@irt1 (dia)	[-0.50,2.50)		▒		
irt2 (dia)			▒	▒	
irt2 (dia)	[93.00,477.00)		▒	▒	▒
age@irt2 (dia)	[8.50,67.50)		▒	▒	
swetna1 (dia)			▒	▒	
swetna1 (dia)	[47.50,80.50)		▒	▒	
swetna1 (dia)	[85.50,1.24E3)		▒	▒	▒
swetna1 (dia)	[80.50,85.50)			▒	
swetk1 (dia)					▒
swetk1 (dia)	[31.50,105.00)				▒
swetk1 (dia)	[20.50,31.50)		▒		▒
swetcl1 (dia)			▒	▒	▒
swetcl1 (dia)	[86.50,113.00)		▒	▒	▒
swetcl1 (dia)	[-11.00,86.50)			▒	
swetna2 (dia)			▒	▒	
swetna2 (dia)	[60.50,139.00)		█	▒	
swetk2 (dia)					
swetk2 (dia)	[13.50,32.50)		▒	▒	
swetcl2 (dia)					
swetcl2 (dia)	[85.50,153.00)		▒	▒	▒
SOURCE (cul)	Sputum		▒	▒	▒
SOURCE (cul)	ThroatCulture		▒	▒	▒
tobraresistent? (cul)					▒
tobraresistent? (cul)	NotDone				▒
tobraresistent? (cul)	No		▒	▒	▒
ciproresistant? (cul)					▒
ciproresistant? (cul)	NotDone				▒
ciproresistant? (cul)	No		▒	▒	▒
meropenemresistant? (cul)					▒
meropenemresistant? (cul)	NotDone				▒
meropenemresistant? (cul)	No		▒	▒	▒
WAZ (per)			▒	▒	▒
WAZ (per)	[-1.75,1.36)		█	█	▒
HAZ (per)			▒	▒	▒
HAZ (per)	[-1.53,2.53)		█	█	█
BASEpower (cul)					▒
BASEpower (cul)	[-0.50,1.15E3)		▒	▒	█
BASEpower2 (cul)					▒
BASEpower2 (cul)	[-1.75,752.00)		▒	▒	█
BASEpower3 (cul)					▒
BASEpower3 (cul)	[-0.50,5.5E7)		▒	▒	█
BASEpower4 (cul)					▒
BASEpower4 (cul)	[-2.5E3,2.5E3)		▒	▒	▒
age@diagnosis (dia)			▒	▒	▒
age@diagnosis (dia)	below025		▒	▒	▒
age@diagnosis (dia)	2till7				▒