45073 Computer Science, Projects

Discretization of Numerical Attributes

Preprocessing for Machine Learning

Stud. Techn. Knut Magne Risvik

Knowledge Systems Group
Department of Computer and Information Science
Norwegian University of Science and Technology
N-7034 Trondheim, Norway.
E-mail : kmr@idi.ntnu.no

Abstract

The area of Knowledge discovery and Data mining is growing rapidly. A large number of methods is employed to mine knowledge. Several of the methods rely of discrete data. However, most datasets used in real application have attributes with continuously values. To make the data mining techniques useful for such datasets, discretization is performed as a preprocessing step of the data mining.

In this paper we examine a few common methods for discretization and test these algorithms on common datasets. We also propose a method for reducing the number of intervals resulting from an orthogonal discretization by compromising the consistency level of a decision system. The algorithms have been evaluated using a rough set toolkit for data analysis.

In Chapter 2, we introduce Knowledge Discovery. We also discuss the preprocessing step in the Knowledge Discovery pipeline, and introduce discretization in particular.
Chapter 3 introduces some basic notions in Rough Set theory.
In Chapter 4, we further discusses the discretization process, and investigates some common methods for discretization.
In Chapter 5, we propose a two-step approach to discretization, using the Naive discretization algorithm introduces in 4.2, and a proposed algorithm to merge intervals.
Empirical results from comparison of the different algorithms discussed in Chapter 4, and the proposed method from Chapter 5 can be found in Chapter 6.

In Chapter 7, further work in the area of discretization is discussed.

Appendix A contains some notes on the Rosetta[15] framework, and the implementation of the investigated algorithms in particular.

Acknowledgements

First of all, I would like to thank my supervisor, Ph.D. student Aleksander Øhrn for his co-operation and support while performing this project. By being my supervisor he has given me a great introduction to the research field of knowledge discovery and data mining. He also gave me some ideas, that were the basis of some of the proposed algorithms in this report. I would also like to thank Prof. Dr. Jan Komorowski and Ph.D. student Staal Vinterbo for discussing some topics with me.

Also, thanks to my colleagues at the User support service (Orakeltjenesten) for allowing me to use our computers extensively for writing code and performing tests.

Knut Magne Risvik
Trondheim, 29.04.97

Table of contents

Chapter 1 Introduction

This report is the results of the work performed in the course 45073 Computer Science, Projects at the Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU), Trondheim, Norway.

The course is compulsory for all 4th year computer science students at the department.

1.1 Motivation

Tremendous amounts of data is being sampled and collected each day. Much of these data are of no direct use. Intelligent automated techniques are needed to extract knowledge from all this data.

Several methods exists for knowledge discovery, and the research in this area is extensive from different angles of attack. Several of the traditional fields of Artificial Intelligence (AI) research can be applied to this area, combined with new ideas from pattern recognition and statistics.

Combining all these areas of research is a great challenge to the Knowledge discovery community. This challenge, along with the enormous capabilities of such systems, makes this a very exciting area of research.

1.2 Scope

In this report, we will focus on some methods used in preprocessing in knowledge discovery. The total project work is the combination of some theory investigations, along with the implementation of the investigated methods.

We discuss methods for discretization of numerical attributes. We limit ourself to investigating methods yielding orthogonal decision areas. The most commonly referred algorithms is discussed and compared on some datasets.

Chapter 2 Knowledge Discovery

Data are being collected and accumulated at dramatically high rates across a wide variety of fields. The need for computational tools to extract useful information from the tremendous amount of data is rapidly growing. Theories and tools for such extraction are subject of the field of Knowledge discovery in Databases (KDD).
An informal definition of KDD is the methods and techniques used to make sense of the data. Most sampling processes return a huge amount of data, way to large to digest and interpret manually. The main goal of KDD is to reduce the amount of data, while preserving the knowledge that the original set of data represents.

2.1 The Knowledge Discovery process

Fayyad, Piatetsky-Shapiro and Smyth [1] formalizes the definition of the KDD process :

> Knowledge Discovery in Databases is the non-trivial *process* of identifying *valid*, *novel*, *potentially useful*, and *ultimately understandable patterns* in data.
> Data is a set of facts. (e.g., cases in a database). ■

where *pattern, process, validity, novel, potentially useful* and *ultimately understandable* are defined as follows.

Pattern: An expression E in a language L describing facts in a subset F_E of F. E is called a pattern if it is simpler (in some sense) than the enumeration of all facts in F_E.

Process: Usually the KDD process is a multi-step process, which involves data preparation, search for patterns, knowledge evaluation, and refinement involving iteration after modification. The process is assumed to be non-trivial - that is, to have some degree of search autonomy.

Validity: The discovered patterns should be valid on new data with some degree of certainty. A measure of certainty is a function C mapping an expression in L to a partially or totally ordered measurement space M_C. An expression E in L about a subset $F_E < F$ can be assigned a certainity measure $c = C(E,F)$.

Novel: The patterns are novel (at least to the system). Novelty can be measured with respect to changes in data (by comparing current values to previous or expected values) or knowledge (how a new finding is related to old ones). In general, we assume this can be measured by a function $N(E,F)$, which can be a Boolean function, or a measure of degree of novelty or unexpectedness.

Potentially Useful: The patterns should potentially lead to some useful actions, as measured by some utility function. Such a function U maps expressions in L to a partially or totally ordered measure space M_U: hence, $u = U(E,F)$.

Ultimately Understandable: A goal of KDD is to make patterns understandable to humans in order to facilitate a better understanding of the underlying data. While this is difficult to measure precisely, one frequent substitute is the simplest measure. Several measures of simplicity exists, and they range from the purely syntactic (e.g., the size of a pattern in bits) to the semantic (e.g., easy for humans to comprehend in some setting). We assume this is measured, if possible, by a function S mapping expressions E in L to a partially or totally ordered measure space M_S: hence, $s = S(E,F)$.

The process of Knowledge discovery is a multi-step process, and it can be described as a pipeline refining raw data into knowledge. Figure 1 illustrates the different steps in the KDD process.

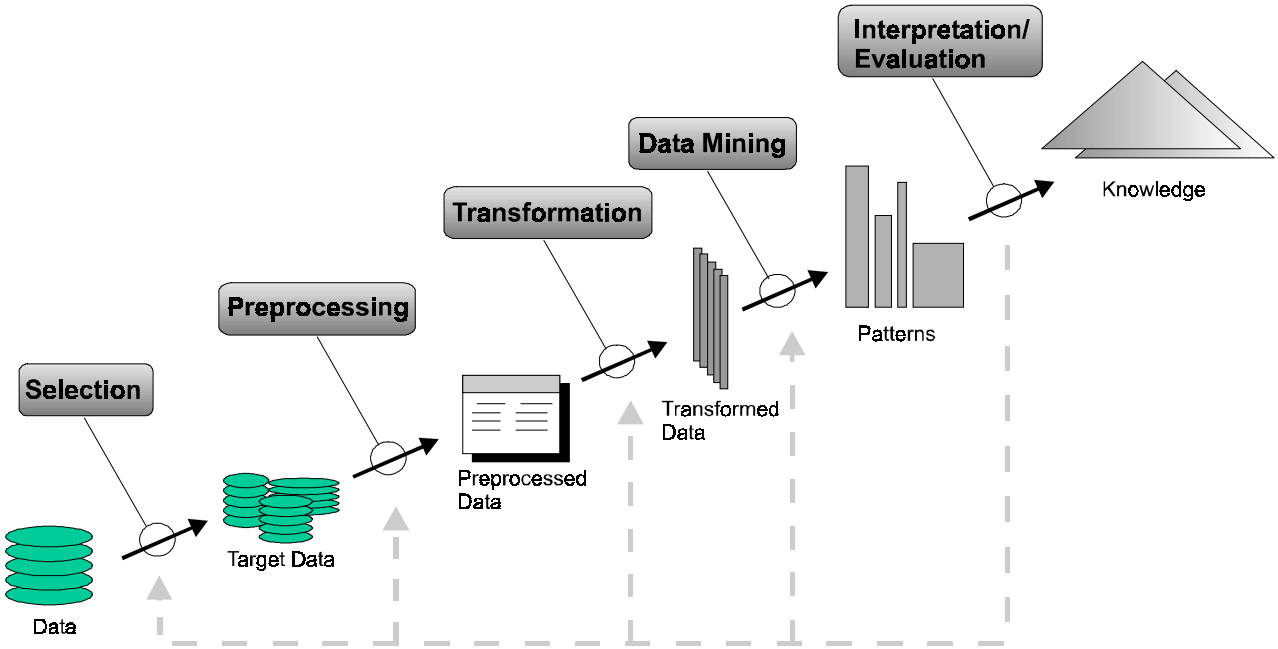


Figure 1 Overview of the steps comprising the KDD process

This report is concerned with the **Preprocessing** step of the KDD process, and is further limited to *discretization*.

2.2 Discretization

When operating on objects with continuous or numerical attributes, knowledge of singleton values have limited usefulness on new objects. Discretization alters the granularity of how an attribute's value is recorded, in effect "coarsening" our perception of the world.

Definition (Discretization). Discretization is a function:

$$Q : D \rightarrow C \qquad \textbf{(2.2.1)}$$

assigning a class $c \in C$ to each value $v \in D$ in the domain of the attribute being discretized. A desirable feature for a practical discretization is:

$$|D| > |C| \qquad \textbf{(2.2.2)}$$

I.e. , a discretized attribute have fewer possible values.

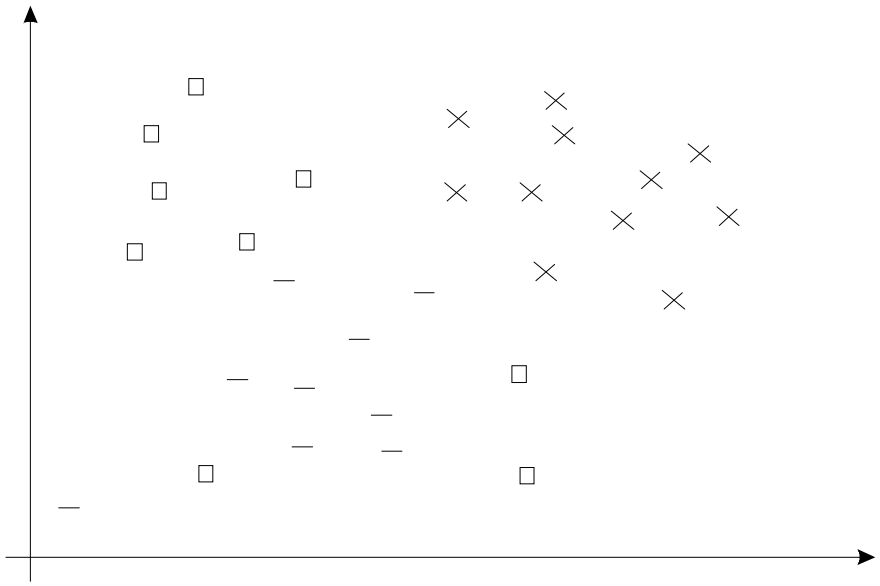


Figure 2 Distribution of objects with two attributes

The two-dimensional case in Figure 2, denotes objects with two attributes. The objects are classified on the basis of these two attributes, letting dashes, squares and crosses denote the different classes of objects.

Obviously, we would like to get knowledge that is **valid** (in the sense of Fayyad et.al.[1] for new objects, that may have different attribute values than any of the objects already found in the system.

To achieve this, being able to find some areas of classification would be of help.

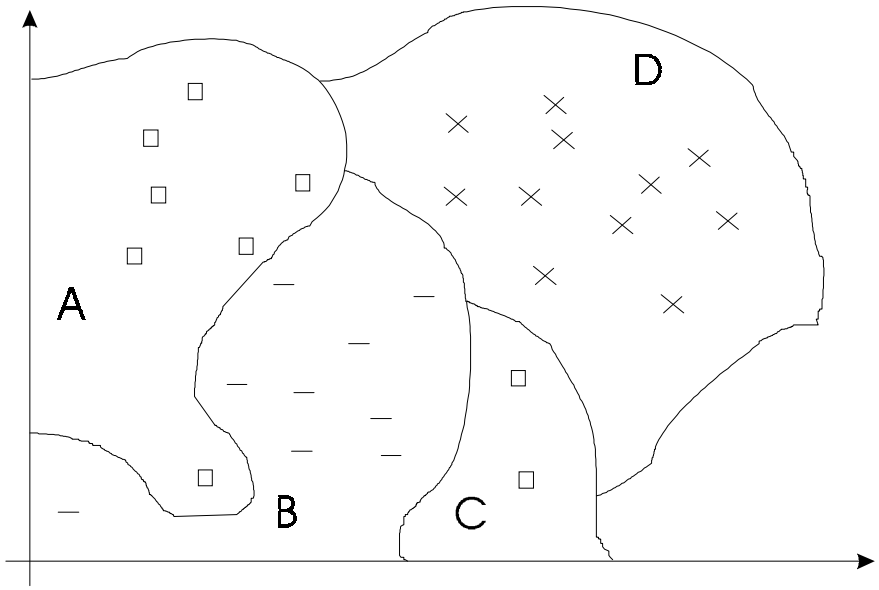


Figure 3 General classification areas

In Figure 3 we have drawn decision boundaries of object classification. Denoting that all objects that reside within area *A* or area *C* have the same classification as the square objects, objects in area *B* have the dash classification, and objects in area *D* are classified as crosses.

Having found such decision boundaries, tremendously improves the **validity** (in the KDD sense) of the knowledge, since we now have knowledge that covers a much large universe of objects.

The boundaries shown in Figure 3 are general in terms of their geometric properties. Finding such boundaries also means that some dependency between the attributes have been found. The boundary can be described as a parametric closed curve in a *n*-dimensional affine space. Finding such dependencies means altering the immediate semantics of the data. Since there is no information on the semantics of the data in the KDD process, the computing machinery has no way of knowing if the proposed dependency makes sense.

If we let the decision boundaries be described as linear functions (i.e. they are hyperplanes), they may look like:

$$(0.05*\mathit{Bloodpresure}+.123*\mathit{Haircolor}) > 12.25 \rightarrow \mathit{Heartcondition(BAD)}$$

Such rules certainly make little sense to other than a expert human observer. Such a rule may, however, be found as a result of searching for areas in the object space. Linear functions are again a special case of more general areas. Several methods exists that make use of linear dependencies or other techniques that alters the original semantics of the attributes.

Methods using general hyperplanes have been proposed, for instance by Skowron, Hoa and Son[2]. A method based on clustering techniques can be found in for example Rauber, Coltuc and Garcao[3].

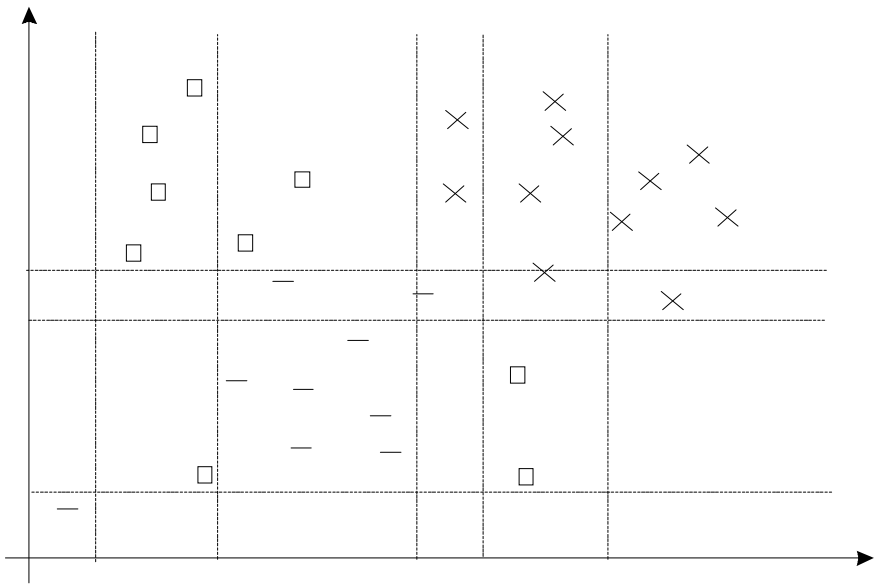


Figure 4 Orthogonal classification areas

Limiting the form of the areas even more, results in orthogonally defined areas, i.e. areas limited by lines that are parallel to the axes. This is illustrated in Figure 4, where each axis can be divided into certain intervals for which a classification is true.
Using this type of areas, we can find rules of the form:

- If $V_1 \leq a < V_2$ **AND** $V_3 \leq b < V_4$ then classification is $\mathbf{C_1}$
- If $V_4 \leq c < V_5$ **AND** $V_6 \leq d < V_7$ then classification is $\mathbf{C_2}$

To illustrate this, we introduce a simple *decision system* (Table 2.2.1) to classify vehicles based on their length illustrates the problem with attributes of a continuous nature.

	Attribute	
Object	**Length [m]**	**Class**
1	5.30	Car
2	4.25	Car
3	14.53	Bus
4	12.69	Bus

Table 2.2.1 - A simple decision system with continuos attributes

Using a Data mining tool to mine rules for this decision system (The notion of a *decision system* will be defined more firmly in Chapter 3) could for example yield the following rules :

- *Length(***5.30***)* → *Class(**Car**)*
- *Length(***4.25***)* → *Class(**Car**)*
- *Length(***14.53***)* → *Class(**Bus**)*
- *Length(***12.69***)* → *Class(**Bus**)*

Of course, these rules will yield correct results for the objects in Table 2.2.1, but for classifying new objects these rules will be useless since the likehood of getting a new object with a length that equals one of the lengths that the rules support is very small. So these rules are not **valid** (recall definition from chapter 2.1) for any new objects.

Applying a discretization algorithm as a preprocessing step for the data mining, we might get the following rules:

- $Length < \mathbf{9.00} \rightarrow Class(\boldsymbol{Car})$
- $Length \geq \mathbf{9.00} \rightarrow Class(\boldsymbol{Bus})$

Rules on this form, will tell us that vehicles that have length within a given interval is classified as cars, whether vehicles with length within an other interval are classified as buses. Obviously such rules could be much more applicable to unseen objects, than rules that only works for known cases.

As seen from the small example above, the need for some kind of discretization processing is required to generate rules that could prove useful for new cases. The discretization process studied in this report converts continuos attributes into discrete ones, yielding intervals which the attribute value could reside in, instead of singleton values.

Numerous methods exists for discretization, based on different theoretical origins:

- Statistics
- Boolean Reasoning
- Clustering techniques
- Error and Entropy calculations

In this report, we restrict our focus to of the methods for discretization that yields orthogonal classification areas, i.e. cuts on the axes.

Chapter 3 Rough Set Preliminaries

The techniques for discretization discussed in this report are general preprocessing methods, and are not directly related to any particular method for Data mining. However, the data analysis tool that will be used to test some of the algorithms discussed in this report uses Rough Sets to perform analysis. Also, we will refer to the rough set theory in some of the methods discussed. We therefore introduce some basic notions of Rough Sets.

Rough Sets were introduced by Pawlak[4] as a tool to deal with uncertain or vague knowledge in AI applications. Central to the Rough Sets approach is classification; the ability to distinguish between objects , and consequently reason about partitions of the universe. Below we summarize some basic and important concepts and properties regarding rough sets. For a more detailed discussion of Rough Sets, the reader is encouraged to consult [4]. Also, a very good introduction to Rough sets can be found in [5].

3.1 Data Representation

As a starting point we have data collected and represented in an *Information System* containing knowledge about objetcts in terms of a predefined set of *attributes* or *properties.* All objects in the *Information System* have values for these attributes.

Defintion (Information system). An Information System (IS) is an ordered pair $\boldsymbol{A}=(U,A)$, where U is a non-empty set representing the universe of discourse. The non-empty set A is the set of attributes or properties of the objects in the IS. Every attribute $a \in A$ is a function

$$a : U \rightarrow V_a \qquad \textbf{(3.1.1)}$$

where V_a is the range of allowable values for attribute a. If the function is a total function, we have no missing values in the IS. ■

	Attribute		
Object	**Color**	**Shape**	**Eatable**
1	White	Spherical	No
2	Green	Spherical	Yes
3	Yellow	Spherical	Yes
4	White	Spherical	No
5	Yellow	Square	Yes

Table 3.1.1 An example IS, presented as a table

A typical example of an information system is a relational database table as shown in **Error! Reference source not found.**. This IS describes three different properties on five different objects.

Some Information Systems have a special attributes which denotes decisions based on the other attributes (or other properties of the object not visible in the IS). An Information System with such decision attributes is called a *Decision System.*

Definition (Decision System). A decision system is an Information system $\boldsymbol{A}=(U,A \cup \{d\})$ where the d is called the decision attribute, and $d \notin A$. ■

	Attributes			
Object	**Color**	**Shape**	**Eatable**	**Classification**
1	White	Spherical	No	White Rock
2	Green	Spherical	Yes	Apple
3	Yellow	Spherical	Yes	Orange
4	White	Spherical	No	Football
5	Yellow	Square	Yes	Cheese

Table 3.1.2 Sample decision system

Adding a new column to **Error! Reference source not found.**, called *classification*, yields us with a decision system that classifies objects, preferable on basis of the attributes in the table. The decision system is shown in **Error! Reference source not found.**.

3.2 Indiscernibility

Central in the Rough Set theory is the concept of discernibility. For two objects with different decision attribute values, we would like to be able to discern them, based on the values of the condition attributes. For a subset of the attributes (or the whole set of attributes) this may however not be achievable. An indiscernibility relation relates objects, which, for a given subset of the attributes *A*, are indiscernible.

Definition (Indiscernibility Relation). For every subset of attributes $B \subseteq A$ in the IS $\boldsymbol{A}=(U, A)$, we can associate a relation called the *Indiscernibility Relation*, which can be defined as follows:

$$IND(B)=\{(x, y) \in U^2 \mid \forall a \in B, a(x) = a(y)\} \quad \textbf{(3.2.1)}$$

U/IND(B) denotes the set of all equivalence classes in the relation IND(B). ■

Selecting a subset B of attributes from A, partitions the universe into *classes* of objects that are indiscernible by attributes in B.

In the following definitions will be given in terms of *object classes*, which are to interpreted as the classes $E_i \in U$/IND(B) induced by a set of attributes $B \subseteq A$.

For the example decision system in Table 3.1.2, the set of classes for the set $B=A$ is :

$$U / IND(B) =\{\{1,4\},\{2\},\{3,5\}\} \quad \textbf{(3.2.2)}$$

A *discernibility matrix* can be used to show all attributes that discern all pair of objects in the Universe of discourse.

Definition (Discernibility Matrix). For a set of attributes $B \subseteq A$ in $\boldsymbol{A}=(U,A)$, the *Discernibility matrix* $M_D(B) = \{m_D(i,j)\}_{nxn}$, $1 \le i, j \le n = |\ U/\text{IND}(B)\ |$, where

$$m_D(i, j) = \{a \in B \mid a(E_i) \neq a(E_j)\}, i, j = 1,2,\ldots,n \quad \textbf{(3.2.3)}$$

The entry $m_D(i,j)$ in the discernibility matrix is the set of attribute from B that discerns object classes $E_i, E_j \in U$/IND(B). ■

Now, we associate a unique Boolean variable • . Hence, to each element $m_D(i,j)$ in the discernibility matrix corresponds a set

$$\overline{m}_D(i, j) = \{\overline{a} \mid a \in m_D(i, j)\} \quad \textbf{(3.2.4)}$$

Definition (Discernibility function). The *Discernibility Function f(B)* of a set of attributes $B \subseteq A$ is

$$f(B) = \bigwedge_{i, j \in \{1..n\}} \bigvee m_D(E_i, E_j) \quad \textbf{(3.2.5)}$$

where n = | U/IND(B) |, and the disjunction is taken over the set of Boolean variables that corresponds to the discernibility matrix elements $m_D(i,j)$. ■

As an example, we take the subset $B = A$ for the decision system in Table 3.1.2. The discernibility matrix is shown in Table 3.2.1. The matrix is, of course, symmetrical.

	1	**2**	**3**	**4**	**5**
1	Ø				
2	Eatable	Ø			
3	Color, Eatable	Color	Ø		
4	Ø	Color, Eatable	Color, Eatable	Ø	
5	Color, Shape, Eatable	Color,Shape	Shape	Color, Shape, Eatable	Ø

Table 3.2.1 - Discernibility matrix for decision system

3.3 Data Information Reduction

Equivalence classes reduces the dataset in number of objects. However, we would like to introduce some notion for reducing the number of attributes.

From the discernibility function, we find a subset $B \subseteq A$ of attributes that is enough to discern between all objects in the Information system. Other attributes are said to be *superfluous* or *dispensable*.

Definition (Dispensability). An attribute a is said to be *dispensable* or *superfluous* in $B \subseteq A$ if IND(B) = IND(B - {a}), otherwise the attribute is said to be *indispensable* in B. ■

A subset $B \subseteq A$ of attributes, that can discern between all the objects in the Information system, that has some superfluous attributes removed is called a reduct.

Definition (Reduct). A *Reduct* of B is a set of attributes $B' \subseteq B$ such that all attributes $a \in B$-B' are dispensable, and IND(B') = IND(B). The term RED(B) denotes the family of reducts of B. The set of reducts of B is determined by the set of prime implicants of the discernibility function *f(B)*. ■

In some cases, our need to classify objects is limited to discern one class of objects from all other classes of objects. A reduct (set $B \subseteq A$ of attributes) that discerns objects from one class X_i , from all other classes of objects is called a *relative reduct.*

Taking the intersection of all reducts for a set $B \subseteq A$ of attributes, gives us the *core*.

Definition (Core). The intersection of all reducts of a set of attributes $B \subseteq A$, is called the core of B.

$$CORE(B) = \bigcap RED(B) \qquad \textbf{(3.3.1)}$$

■

3.4 Set Approximations

For a set of attributes $B \subseteq A$ and an equivalence class X of objects, we can partition the elements from U into three different sets:

1) The set of objects that certainly are members of X, according to the attributes B.
2) The set of objects that possibly are members of X, according to the attributes B.
3) The set of objects that certainly are not members of X, according to B.

This partition can be formalized into the definition of *Lower Approximation* and *Upper Approximation*.

Definition (Lower and Upper Approximation). The classes

$$\begin{aligned} \underline{B}X &= \bigcup (E \in U / IND(B) \mid E \subseteq X\} \\ \overline{B}X &= \bigcup (E \in U / IND(B) \mid E \cap X \neq \varnothing\} \\ BN_B(X) &= \overline{B}X - \underline{B}X \end{aligned} \qquad \textbf{(3.4.1)}$$

is called the B-lower and the B-upper approximation, respectively. The region $BN_B(X)$ is called the B-boundary region of X. ■

The elements in the boundary region are elements that can neither be classified as being definitely within nor definitely outside X using attributes B. For a given class of objects it is therefore possible to introduce a *membership function*.

Definition (Rough Membership function). For $\boldsymbol{A}=(U,A)$, $x \in U$, $X \subseteq U$, attributes $B \subseteq A$, the *Rough Membership function* for a class $E \in U/\mathrm{IND}(B)$ is

$$\mu_B(E,X) = \frac{|E \cap X|}{|E|}, 0 \leq \mu_B(E,X) \leq 1 \qquad \textbf{(3.4.2)}$$

■

3.5 Decision Function

On basis of the objects in the decision system, we can extract functions that determine the decision value of the object. In some cases, we can have several possible decision values for an object.

Assuming that set V_d of values of the decision attribute d is equal to $\{1,\ldots,r(d)\}$.

Definition (Generalized decision). The function $\partial_A : U \rightarrow \mathbf{P}(\{1,\ldots,r(d)\}$, called the *generalized decision*, is defined by

$$\partial_A(x) = \{i \; \exists x' \in U, x' \, IND(A) x, d(x) = i \qquad \textbf{(3.5.1)}$$

Definition (Inconsistent decision table). A decision table ***A*** is called *consistent (deterministic)* if $card(\partial_A(x))=1$ for any $x \in U$, otherwise ***A*** is inconsistent (non-deterministic).

Chapter 4 Discretization of Data

Several methods have been proposed to discretize data as a preprocessing step for the data mining process. Roughly we can divide the approaches into two classes:

- **Unsupervised methods.** "Blind" methods, where we have no classification information available for the object being considered. These methods rely on assumptions of the distribution of the attribute values.
- **Supervised methods.** Classification information is available, and this information can be taken into consideration when discretizing the data. A common denominator can be to minimize the number of objects from different decision classes into the same discretization class.

We will look into several methods for discretizing data, with the main focus on four simple supervised methods. First we mention the idea behind two unsupervised methods.

We chose to investigate four of the most commonly referred algorithms in the machine learning literature, and we also propose a very simple algorithm ourself.

4.1 Unsupervised methods

We briefly discuss the ideas behind a couple of unsupervised methods.

4.1.1 Equal width discretization

This is the simplest approach to discretization. For each attribute a, calculate min(V_a) and max(V_a). A variable k, will determine the number of intervals preferred. Letting :

$$d = \frac{\max(V_a) - \min(V_a)}{k} \quad \textbf{(4.1.1)}$$

yields the following cuts in the intervals :

$$\min(V_a) + d, \ldots, \min(V_a) + (k-1)d \quad \textbf{(4.1.2)}$$

The equal width method assumes that the underlying data fits somewhat well into a uniform distribution. It is very sensible to outlayers, and can perform extremely bad under some conditions.

4.1.2 Equal frequency discretization

This is also a unsupervised method. Counting the number of values we have from the attribute it will try to partition the attribute into intervals containing the same number of instances.

Both these methods are unsupervised and will not take the decision attributes into consideration. Therefore classification can be made difficult in several cases.

4.2 A Naive Algorithm

Taking the decision into consideration, we can propose a very simple discretization algorithm. Sorting the objects according to the value of the attribute being considered, we can "walk" along the axis of the attribute, creating a new interval whenever the decision value is changing. This algorithm yields a set of all required cuts to keeping the *consistency level* (will be defined more precisely in 5.1) in the information system constant.

Algorithm (NaiveDiscretization).

1. Select attribute $a \in A$.
2. Set $C_a = \emptyset$.
3. Sort values of the attribute.
4. For all adjacent pairs of objects if $V_a(i) \neq V_a(i+1)$, and $d(i) \neq d(i+1)$, create a new cut $c = (V_a(i) + V_a(i+1))/2$, and $C_a = C_a \cup c$.
5. Repeat 2-4 for all continuous attributes a in $\boldsymbol{A}$.

This algorithm is bounded by the sorting step, therefore the computational complexity is $O(k\ N \log N)$, where k is the number of continuous attributes, and N is the number of objects in the decision system.

4.3 The ChiMerge and Chi2 algorithms

The ChiMerge algorithm[6] have been widely used and referred in machine learning papers, and investigating this method came as a natural choice. The algorithm has been refined into a modified Chi2 version[7].

We will first introduce the basic notion of a contingency table, and some definitions of Chi-square statistics.

4.3.1 Chi-square statistics

To classify sample objects according to two different criteria, C_1 and C_2, sample objects can be arranged in a square array, known as a **contingency table**. The table will have *r* rows, one for each class relating to criterion C_1, and *c* columns, one for each class relating to criterion C_2. If a sample of *n* units is to be taken and each unit is to be classified according to the two criteria, we will let N_{ij} denote the number of units occupying both cell *i* of criterion C_1 and cell *j* of criterion C_2. n_{ij} is the observed frequency corresponding to the random variable Nij. These concepts are illustrated in Table 4.3.1, where V_1 and V_2 denotes that a object corresponds to criteria C_1 and C_2, respectively.

	V_1	V_2	
D_1	n_{11}	n_{12}	n_1
…			
D_r	n_{r1}	n_{r2}	n_2
	n_1	n_2	N

Table 4.3.1 - Contingency table

Taking the decision system from Table 2.2.1, we can create two criterions:

- Object has length < 9.00 m
- Object has length ≥ 9.00 m

We also have two decisions, **Car** or **Bus.**

The resulting contingency table is shown in Table 4.3.2

	Criterion		
Decision class	**< 9.00 m**	**≥ 9.00 m**	
Car	0.50	0	0.50
Bus	0	0.50	0.50
	0.50	0.50	1.0

Table 4.3.2 - Sample contingency table for decision system

Given that the sampling process is random, each selection has a probability p_{ij} of being in the *ij*th cell, and the expected frequency is:

$$e_{ij} = E\lfloor N_{ij} \rfloor = np_{ij} \qquad \textbf{(4.3.1)}$$

A measure for closeness between assumed hypothesis and the sample data can be:

$$(N_{ij} - np_{ij})^2 \qquad \textbf{(4.3.2)}$$

Then it can be shown that the random variable:

$$W = \sum_{i=1}^{r} \sum_{j=1}^{c} \frac{(N_{ij} - np_{ij})^2}{np_{ij}} \qquad \textbf{(4.3.3)}$$

possesses a distribution that is approximately chi-square for sufficiently large n.

Given that the classification criteria are independent, each classification corresponds to two random variables, say X_1 and X_2, for each dimension in the contingency table. Letting u_i = P(X_1=i) and v_j = P(X_2=j), independence of the criteria yields :

$$P_{ij} = P(X_1 = i, X_2 = j) = P(X_1 = i)P(X_2 = j)u_i v_j \qquad \textbf{(4.3.4)}$$

for i =1,2,…,r and j = 1,2,…,c. The above chi-square statistics takes the form

$$W = \sum_{i=1}^{r} \sum_{j=1}^{c} \frac{(N_{ij} - nu_i v_j)^2}{nu_i v_j} \qquad \textbf{(4.3.5)}$$

The maximum-likehood estimates of u_i and v_j are :

$$\begin{aligned} \overset{*}{u}_i &= \frac{m_i}{n} \\ \overset{*}{v}_j &= \frac{n_j}{n} \end{aligned} \qquad \textbf{(4.3.6)}$$

The estimate of the probability of being classified in the ith cell relative to C_1 is the number of elements the ith cell (row) divided by the total number in the sample, and the estimate of the probability of being classified in the jth cell relative to C_2 is the number of elements in the jth cell (column) divided by the total number of the sample.

Substituting the corresponding estimators for the above maximum-likehood estimates yields the following χ^2 -sample value:

$$W = \sum_{i=1}^{r} \sum_{j=1}^{c} \frac{(n_{ij} - e_{ij})^2}{e_{ij}} \quad \textbf{(4.3.7)}$$

where:

$$e_{ij} = \frac{m_i n_j}{n} \quad \textbf{(4.3.8)}$$

that is compared to the critical value.

When to intervals are considered for merging, the criterion C_2 will be that a test sample corresponds to each of the intervals. The C_1 criterion is the decision value of each test sample. Assuming that we have a binary decision attribute, gives r=2, which in turns yields a formula used in both the ChiMerge and Chi2 algorithms:

$$\chi^2 = \sum_{i=1}^{2} \sum_{j=1}^{k} \frac{(A_{ij} - E_{ij})^2}{E_{ij}} \quad \textbf{(4.3.9)}$$

The *ChiMerge* method was described by Kerber[6], and is an supervised method. Kerber defines a criterion for measuring the quality of a discretization, stating *"in an accurate discretization the relative class frequencies should be fairly consistent within an interval, but two adjacent intervals should not have similar relative class frequencies"*. ChiMerge uses χ^2-statistics to determine the independence of the class from two adjacent intervals, combining them if it is independent, else letting the intervals be separate.

Algorithm (ChiMerge).

The algorithm can be described in the following steps :

1. Sort the dataset by the attribute being discretized. Initialize the discretization by putting each object into its own interval.

2. Compute χ^2- value for each pair of adjacent intervals, using the following formulae :

$$\chi^2 = \sum_{i=1}^{2} \sum_{j=1}^{k} \frac{(A_{ij} - E_{ij})^2}{E_{ij}} \quad \textbf{(4.3.10)}$$

Where :
k = number of classes
A_{ij} = number of examples in ith interval, jth class.
R_i = number of objects in the ith interval

C_j = number of examples in the jth class
N = total number of examples
E_{ij} = expected frequency of A_{ij} = $(R_i * C_j)/N$

3. Select the pair with the lowest χ^2-value and merge this pair into one interval.
4. Repeat Step 2 - Step 3 until all χ^2-values are above some χ^2-threshold, α. The threshold is found by looking up a desired significance level in a χ^2-table.

■

Immediate complexity is O(N^2), but with some optimizations, the complexity can be reduced to O(N log N). N is the number of examples in the dataset being discretized.

Liu and Setino[7] used the *ChiMerge* algorithm as a base for their *Chi2* algorithm. Specifying a proper value for α can in many cases be difficult. It would therefore be ideal if the data itself could determine α. The *Chi2* algorithm enhances *ChiMerge*, so that the value of α is calculated based on the training data itself. The calculated value of α differs from attribute to attribute, so that it will only continue merging intervals on those attributes where it is needed.

Algorithm (Chi2 Discretization).

The algorithm can be sectioned into two different phases :

Phase 1:
Repeat step 1. - 4. for all continuos attributes.
1. Initialize significance level to a high value (for example 0.5)
2. Sort training data based on attribute
3. Create an interval for each instance of the training data, and initialize the intervals.
4. Calculate χ^2-value for each pair of adjacent interval.
5. Select the pair with the lowest χ^2-value and merge this pair into one interval.
5. Repeat 2. - 4. while intervals can be merged (some χ^2-value is below the threshold).
6. Repeat step 2 - step 5 for all continuos attributes
7. Decrease significance level.
8. Repeat Phase 1 until inconsistency in data exceeds a rate δ.

Phase 2:
1. Starting with the lowest significance level of phase 1, associate each attribute with the level.
2. Sort training data based on attribute
3. Calculate χ^2-values for all adjacent intervals of the attribute.
4. Select the pair with the lowest χ^2-value and merge this pair into one interval.
5. Repeat step 2 - 4 while intervals can be merged (some χ^2-value is below the threshold).
6. If inconsistency in training data does not exceed predefined rate δ, decrease significance level for this attribute, else mark attribute as non-mergeable.
7. Repeat 2. - 7. for all mergeable attributes, until no attributes can be merged.

■

The first phase of the algorithm can be recognized as a generalization of the *ChiMerge* algorithm. Instead of a predefined significance level, *Chi2* provides a wrapping that automatically increments the threshold (decrementing the significance level). Consistency checking is utilized as a stopping criterion. These enhancements ensures that the *Chi2* algorithm automatically determines a proper threshold, still keeping the fidelity of the original data.

The second phase refines the intervals. If any intervals of any of the attributes can be further merged (i. e by decreasing the significance level for that particular attribute) without increasing inconsistency of training data above the given limit, this is done. While the first phase works on a global significance level δ, the second phase uses separate local significance levels for each attribute.

4.4 Entropy-based discretization

Discretization based on entropy criteria have been discussed in several papers, and will be investigated in this section.

Entropy discretization is a supervised method, proposed by Fayyad and Irani[8], also discussed in [9]

4.4.1 Minimum Description Length Principle

The entropy-based approaches utilizes the Rissanens *Minimum Description Length Principle* (MDLP) as a stopping criterion for the recursive algorithm.

Definition (Minimum Description Length Principle). Given a set of competing hypothesis and a set of data, *S*, the MDLP calls for selecting the hypothesis HT for which *Mlength*(**HT**)+*Mlength*(**S**|**HT**) is minimal among the set of hypothesis. *Mlength* denotes the length of the minimal possible encoding of **HT**, while *Mlenght*(**HT**|**S**) is the length of the minimal encoding of the data given the hypothesis. ■

In the entropy-based discretization procedure, MDLP is employed as follows :
Dividing the system into a sender and a receiver, both with identical copies of the set of training cases. However, the receiver lacks the classification information. The sender must transmit this missing information to the receiver. This is done by transmitting a theory along with exceptions to this theory. The MDL principle states that the optimum division between a simple theory with a large number of exceptions and a complex theory with a small number of exceptions is the case when the number of bits required to encode both the theory and the exceptions are minimized. The MDL principle is used as a stopping criterion only. When splitting an interval increases the number of bits needed to encode the theory and exceptions the recursive splitting procedure is ended.

4.4.2 Discretization algorithm

Entropy discretization first sorts all the examples by the attribute being discretized. Afterwards, a recursive divide and conquer approach is used to create the discretization. The algorithm can be described as the following steps :

Algorithm (EntropyDiscretization).

1. Choose the best cutpoint according to the entropy criteria (T_a)
2. Evaluate if the cutpoint is significant according to the MDL principle
   If it is not significant return.
   else recursively call the discretization algorithm for each of the intervals split by the cutpoint T_a.

■

The essential part of this algorithm is step 1. The entropy is a measure of the degree of randomness of the distribution of instances over positive and negative classes.

Defining a contingency table (see 4.3.1) relating to two independent criteria C_1 and C_2. For criterion C_1 there exists two columns V_1 and V_2 which represents the two possible intervals considered. Criterion C_2 is the decision attribute. For each decision $D_1,\ldots D_r$ there is exists a row in the table. As illustrated in Table 4.3.1, we define:

- n_{ij} is the number of observed cases with attribute value V_j and class D_i.

The row total and the column total are defined as :

$$ni = \sum_{j=1}^{2} n_{ij}$$
$$nj = \sum_{i=1}^{k} n_{ij}$$
$$p_{ij} = \frac{n_{ij}}{N} \qquad \textbf{(4.4.1)}$$
$$p_i = \frac{n_i}{N}$$
$$p_j = \frac{n_j}{N}$$

p_{ij}, p_i and p_j are the approximate probabilities from the decision system.

Using these definitions we further define the following entropies :

$$H_D = -\sum_{i=1}^{r} p_i \log_2 p_i \qquad \textbf{(4.4.2)}$$

H_D is the entropy if the decision classes. It is a measure for the degree of randomness of distribution of the decision classes.

$$H_C = -\sum_{j=1}^{2} p_j \log_2 p_j \qquad \textbf{(4.4.3)}$$

H_C the entropy of the attribute itself. H_C is a measure for the information content of the attribute.

$$H_{DC} = -\sum_{i=1}^{2}\sum_{j=1}^{r} p_{ij} \log_2 p_{ij} \qquad \textbf{(4.4.4)}$$

H_{DC} is the entropy of the join events class - attribute value.

$$H_{D|C} = H_{DC} - H_C \qquad \textbf{(4.4.5)}$$

$H_{D|C}$ is the entropy of the classes given the value of the attribute.

A simple measure being used could be H_C. The cutpoint would partition the contingency table into two new tables, with a single column, each table representing a new decision system for each of the two intervals defined by the cutpoint.
The discretization algorithm would then select the cutpoint that minimizes the sum of entropies for the contingency tables.

As pointed out by Quinlan[10][11] this measure tends to favors the intervals with the larger number of values. Therefore a ratio-measure is often used, yielding the following measure for Entropy :

$$E(T_a) = \frac{|I_1|}{|I|} H_C(I_1) + \frac{|I_2|}{|I|} H_C(I_2) \qquad \textbf{(4.4.6)}$$

where $|I_1|$, $|I_2|$ and $|I|$ denotes the cardinality of objects within each of the two intervals, and the cardinality of the total set of objects, respectively.

The stopping criterion, according to the MDLP, is found to be:

$$\begin{aligned} &Gain(A,T;I) < \frac{\log_2(N-1)}{N} + \frac{\Delta(A,T,I)}{N} \\ &where: \\ &Gain(A,T;I) = E(I) - E(T) \\ &and \\ &\Delta(A,T;I) = \log_2(3^k - 2) - (kE(I) - k_1E(I_1) - k_2E(I_2)) \end{aligned} \qquad \textbf{(4.4.7)}$$

k_1 and k_2 denotes the decision classes found in I_1 and I_2 respectively.

4.5 Orthogonal Hyperplanes

Skowron and Nguyen[12] proposed a method for discretization based on Rough Sets and Boolean reasoning.

Orthogonal hyperplanes can be seen as a special case of linear discriminants, used in pattern recognition.

Defining the problem of discretization of real value attributes is done as follows in this approach.

Let $\mathbf{A} = (U, A \cup \{d\})$ be a decision table where $U = \{x_1, x_2, \ldots, x_n\}$. Assuming $V_a = [I_a, r_a) \subset R$ for any $a \in A$ where R is the set of real numbers.

Assume now that the **A** is a consistent decision table.

Let $\mathbf{P}_a$ be a partition of V_a (for $a \in A$) into subintervals, i.e.

$$P_a = \{[p_a^0 = l_a, p_a^1), [p_a^1, p_a^2), \ldots, [p_a^k, p_a^{k+1} = r_a)\}$$
where
$$V_a = [p_a^0 = l_a, p_a^1) \cup [p_a^1, p_a^2) \cup \ldots \cup [p_a^k, p_a^{k+1} = r_a) \qquad \textbf{(4.5.1)}$$
and
$$p_a^0 < p_a^1 < p_a^2 < \ldots < p_a^k < p_a^{k+1}$$

Any $\mathbf{P}_a$ is uniquely defined by the *set of cuts on* V_a: $\{ p_a^1, p_a^2, \ldots, p_a^k\}$ (empty if $card(\mathbf{P}_a) = 1$). The set of cuts on V_a defined by $\mathbf{P}_a$ can be identified by $\mathbf{P}_a$. A family $\mathbf{P} = \{\mathbf{P}_a : a \in V_a\}$ of partitions on **A** can be represented in the form

$$\bigcup_{a \in A} \{a\} \times P_a \qquad \textbf{(4.5.2)}$$

Any $(a, v) \in \mathbf{P}_a$ will be also called *a cut on* V_a.
Then the family $\mathbf{P} = \{\mathbf{P}_a : a \in V_a\}$ defines from $\mathbf{A} = (U, A \cup \{d\})$ a new decision table

$$\mathbf{A}^P = (U, A^P \cup \{d\}) \qquad \textbf{(4.5.3)}$$

Where $A^P = \{a^P : a \in A\}$ and $a^P(x) = i \Leftrightarrow a(x) \in [p_a^i, p_a^{i+1})$ for any $x \in U$ and $i \in \{0,\ldots,k\}$. The table $\mathbf{A}^P$ is called the **P**-*quantization of* **A**.

Any partition **P** defines a valuation val_P of Boolean variables of the form $p(a,k)$ by $val_P(p(a,k)) =$ **true** iff there exists a cut $(a,p_a) \in \mathbf{P}$ satisfying $v_a^k \leq p_a < v_a^{k+1}$. Instead of $val_P(p(a,k))$ we will also write $\mathbf{P} \models p(a,k)$.

Defining the initial set of cuts :

$$A_1 = \bigcup_{a \in A} \{c_i^a = \frac{v_i^a 0 v_{i+1}^a}{2}, 1 \le i \le card(A) - 1\} \quad \textbf{(4.5.4)}$$

For a given subset $X \subseteq U$, and a given cut $c_m{}^a \in A_1$ we further define

- $W^X(c_m{}^a)$ = number of pairs of objects from X discerned by $c_m{}^a$.
- $W_{\mathbf{P}}(c) = W^{X1}(c) + W^{X2}(c) + \ldots + W^{Xm}(c)$, $X_1,\ldots,X_n$ is the equivalence classes of $IND(A^{\mathbf{B}})$.

Based on these definitions Nguyen and Nguyen[13] further proposes an approximate algorithm:

Algorithm (Discretization).

1. **P** = Ø. L = {*U*}; A_1= *initial set of cuts on* **A.**
2. For c ∈ A_1 do *compute (*$\boldsymbol{W_P}(c)$);
3. *Choose* c_{max} *with the largest value* $\boldsymbol{W_P}(c_{max})$ among cuts from A_1 *and set* **P** = **P** ∪ {c_{max}}; $A_1 = A_1 \backslash \{c_{max}\}$;
4. For $X \in$ **L** do
   If c_{max} cuts the set X into X1, X2 then
   Remove X from **L***; Add to* **L** *the two sets* X_1, X_2*;*
5. If *all set from* **L** *consists of objects from one decision class only* then Stop else go to 2.

P is the semi-minimal set of cuts.

The algorithm needs time of order O(kb(|**P**| log n)). Proof can be found in [13].

Chapter 5 Postprocessing of discretized data

Some datasets have very significant clusters of data. Many of the algorithms described in Chapter 4 identifies a large number of clusters, instead of just the most significant clusters.

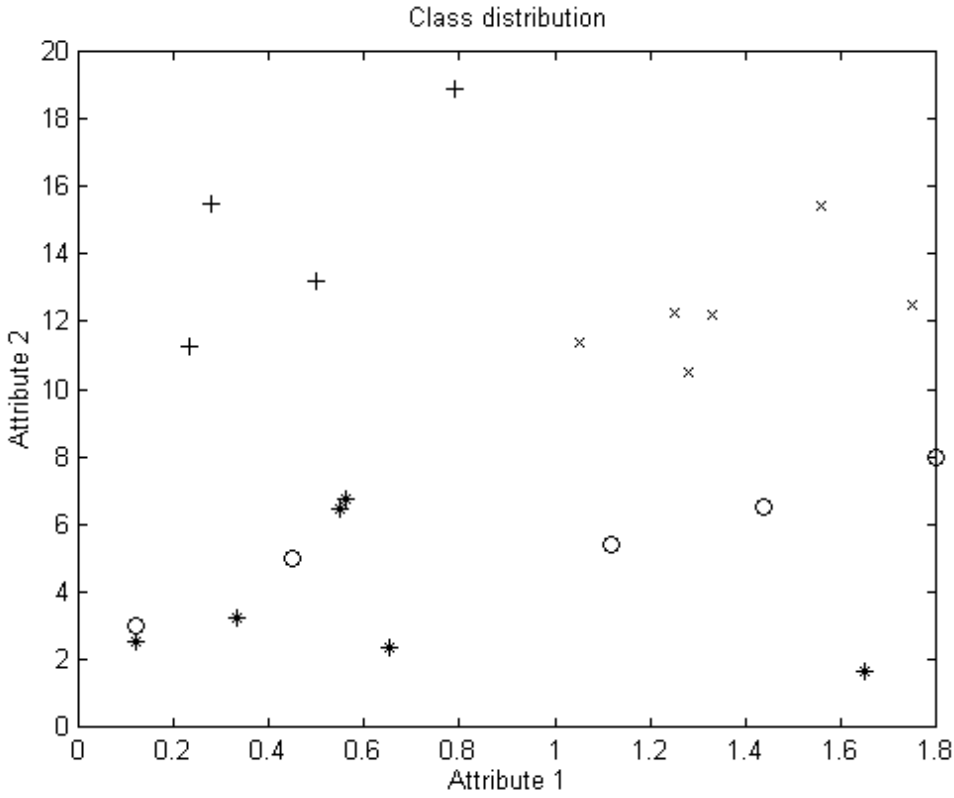


Figure 5 Class distribution in two dimensional Information System

Four quite distinct clusters arise from this picture, and we clearly can see that one cut is required for each attribute to separate objects into the four distinct classes. However, this introduces some degree of inconsistency in the decision system. When operating with a decision system of large cardinality, some degree of inconsistency can be allowed.

Analyzing the data in Figure 5, we can easily identify four major clusters. Based on these clusters, we can find one cut for each of the attributes that roughly gives us the clustering we perhaps would like to have.

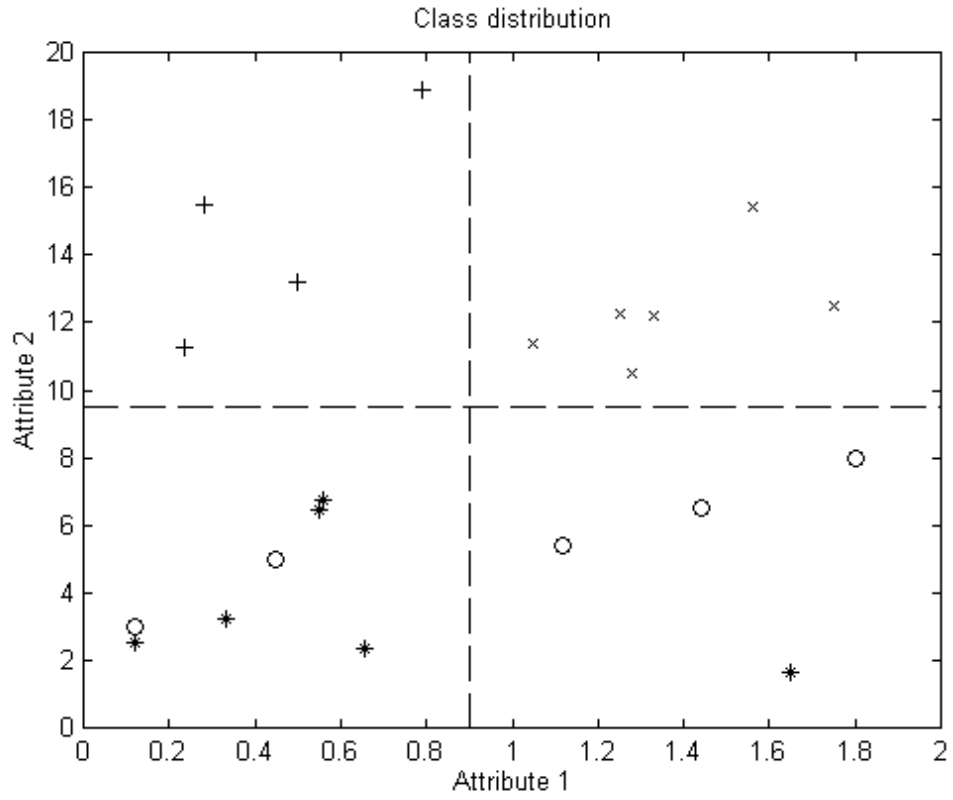


Figure 6 Clustering data with orthogonal cuts

Figure 6 shows the most significant cuts (approximate), and it also points out the objects that introduce inconsistencies in the decision system.

5.1 Data inconsistency

Recalling Equation (3.5.1), and that the decision table is consistent decision table iff $card(\partial_A(x))=1$ for any $x \in U$. Extending this definition, we can define the inconsistency level in a decision table.

Definition (Inconsistency Level). Letting D(B) be the set of decision functions for the attribute set $I \subseteq A$, in $\boldsymbol{A}$, we define the inconsistency level α, as :

$$\frac{\sum_{D(B)} card(\partial_B(x))}{card(D(B))} \qquad \textbf{(5.1.1)}$$

■

Calculating the inconsistency can be done in the following way:

Algorithm (Calculating Inconsistency Level).

1. Create the set of equivalence classes U/IND(B).
2. For each pair of objects within an equivalence class, E_i, do check if they have inconsistent decision values. If so, increase inconsistency level by 1/$card(E_i)$.

5.2 Pruning cuts

Several methods could be proposed to reduce the number of cuts.

1. Removing cuts and merging intervals until a threshold of inconsistency is reached
2. Removing the least significant cuts until a given number of cuts is left.

We will propose an algorithm to perform method 1 above.

Algorithm (IntervalMerger).

1. Select interval attribute $a \in A$.
2. Create equivalence classes $E = U/\mathrm{IND}(a)$.
3. For each pair of adjacent intervals $I_1, I_2 \in A$ calculate increase in decision system inconsistency if they were merged.
4. Select the pair of intervals that gives the lowest increase in decision system inconsistency, and, if the increase does not yield a total inconsistency above threshold, merge the intervals.
5. Repeat Step 1-5 for all interval attribute.
6. Repeat 1-6 until no intervals was merged.

■

This algorithm is not optimal with regards to selection of intervals. The local selection uses a greedy criterion, and does not yield the maximum number of intervals to be merged. A dynamic programming approach to the selection of intervals could improve this. Also, we do not consider all intervals for all attributes when selecting the intervals to merge. This could also easily be achieved.

Since the algorithm chooses to merge intervals that yields the lowest increase in the total inconsistency of the decision system, the algorithm conserves the most significant cuts.

The **Chi2** algorithm[7], described in 4.2, is also based on the concept of merging intervals until a given inconsistency level is reached. Our approach is a generalized edition of this method.

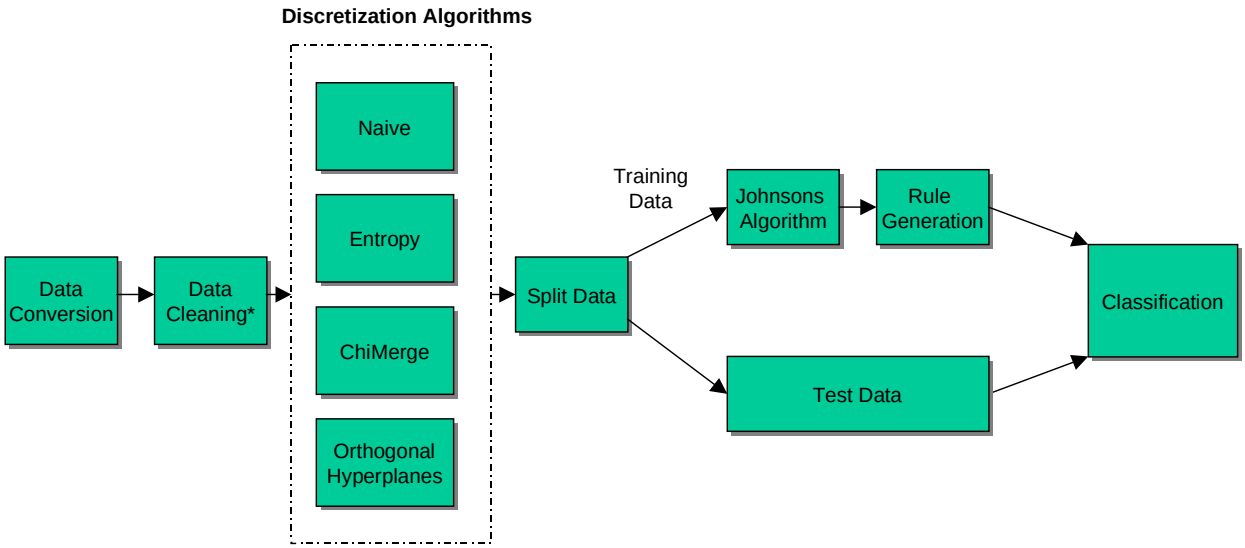
Chapter 6 Empirical Results

Testing was performed to evaluate the performance of the different discretization algorithms described in Chapter 4. We also applied some testing for interval merging algorithm proposed in Chapter 5.

6.1 Comparisons

To compare the different discretization algorithms, we used several datasets with a large percentage of continous attributes.

All data analysis was done with Rosetta[15], the pipeline used to perform the analysis is illustrated in Figure 7.



* Cleaning only applied to Heart dataset

Figure 7 KDD pipeline used in tests

Some of the algorithms required different parameters to be specified. We briefly summarize these parameters:

ChiMerge Discretization algorithm:

- Signficance level = 0.90

Johnsons algorithm for reduct calculation:

- Non object related reducts
- Dynamic Reduct calculation
- 5 sampling levels
- 10 subtables per level
- Smallest subtable 50%, largest 90%

6.1.1 The datasets

To compare the different discretization algorithms we chose three different dataset from the UCI machine learning repository[14]

Iris: The data set contains of 3 classes with 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other. The dataset has 4 continuos attributes.
The dataset was divided into 60 training objects, and 90 testing objects.

Glass: The data set uses the refractive index together with their oxide content for 9 different composites to classify glass into 6 different classes. The dataset have 10 continuously valued attributes, and a total of 214 objects.
The dataset was split into 120 training objects and 94 testing objects.

Heart: The dataset uses several biological and medical attributes (age, sex, bloodpressure, heartrate etc.) to classify objects into four different classes indicating a diagnosis of heart condition. Removing the objects with missing values, gave a total of 299 objects with 13 attributes.
The dataset was split into 272 training objects and 27 testing objects.

6.1.2 Results

The results are presented in Table 6.1.1 as ratios of correctly classified objects from the test sets, using the different discretization algorithms.

Dataset/ Algorithm	Naive Discretization	Entropy Discretization	ChiMerge Discretization	Orthogonal Hyperplanes
Iris	76%	97%	92%	93%
Glass	60%	70%	52%	63%
Heart	44%	59%	44%	67%

Table 6.1.1 Summary of results from comparing different discretization algorithms on selected data sets

We can observe that the results tends to favor the entropy discretization algorithm in most cases. Also, the Orthogonal hyperplanes algorithm seems to give pretty good results in comparison to the other algorithms, at least for the Heart dataset.

There are quite large differences in the performance between the different discretization algorithms. In most cases, the Naive discretization algorithm will produce too many intervals, and the rule generation process will be very sensitive to noisy objects. Reducing the number of interval tends to filter some noisy data, and give rules that are more valid for new objects.

Finding the best confidence level for the ChiMerge algorithm can be difficult, so the results from this algorithm probably can be improved by varying this parameter.

We stress that these results should only be compared to each other. Better results can be achieved for classification of the different datasets. However no published results have a higher correctness ratio for the Iris dataset, than the Entropy Discretization algorithm used for this report.

6.2 Testing interval merging

Two tests were performed with the Interval merging algorithm. One to assure its correctness, and one to test the usefulness of the rules generated after such preprocessing.

6.2.1 Verification of correctness

Starting with the dataset shown in Figure 5, we will first use the naive discretization algorithm (see 4.2) to create all intervals needed to ensure consistency of decision system. We then apply the Interval merging algorithm to reduce the number of intervals.

The initial dataset after applying the Naive Discretization algorithm is shown in Figure 8.

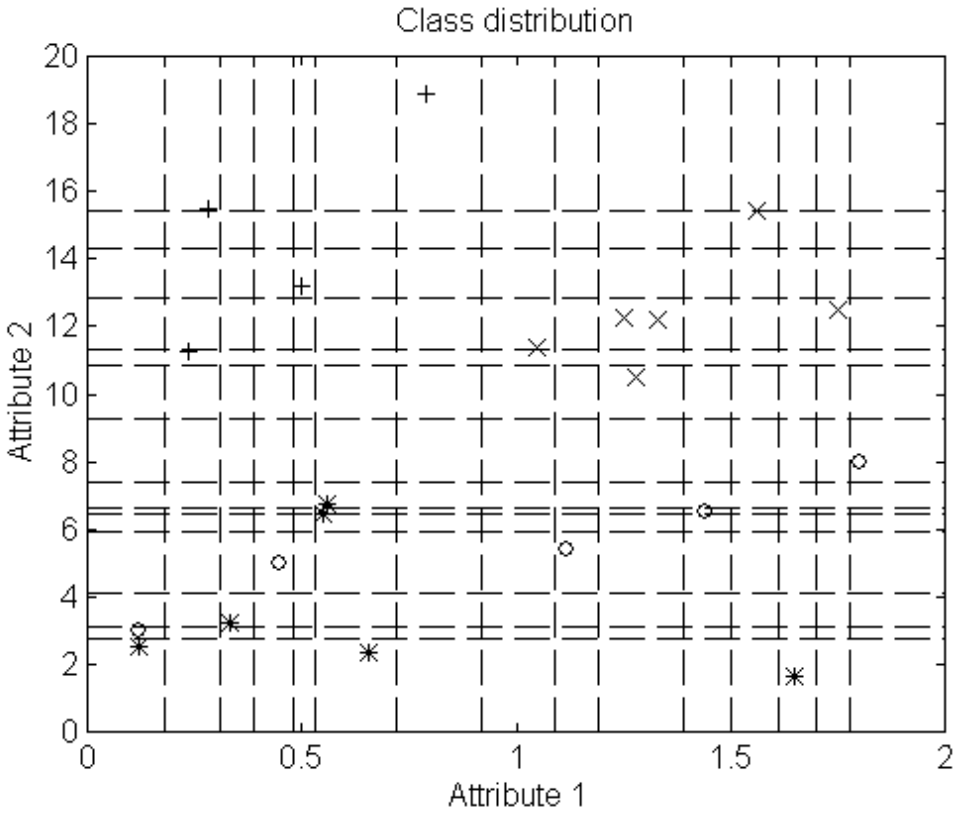


Figure 8 Initial set of interval after applying the Naive discretization algorithm

We thereafter applied the interval merging algorithm with a requested consistency level α = 0.90 to this decision system. The resulting intervals is shown in Figure 9.

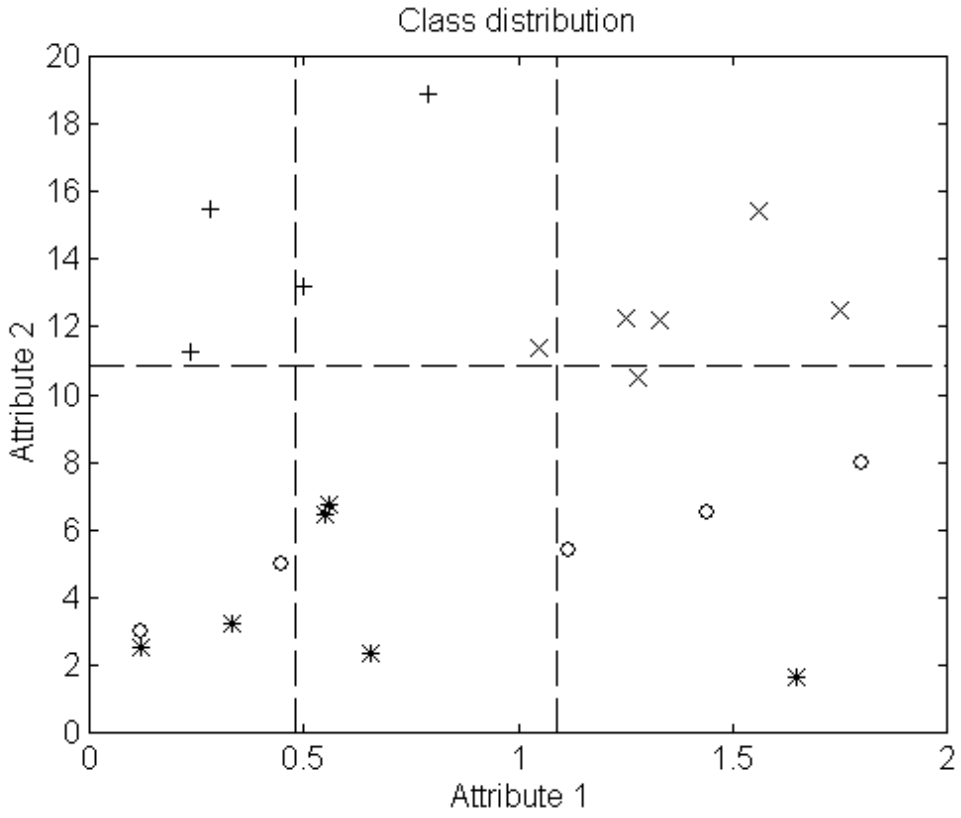


Figure 9 Intervals after applying the interval merge algorithm with consistency level α=0.9.

6.2.2 Testing usefulness

To test the interval merging algorithm usefulness, we applied the algorithm to the Iris dataset. Classification was performed by discretizing the data using the Naive discretization algorithm (4.2), and thereafter applying the interval merging algorithm with different levels α of consistency. The results was compared with the best classification acquired in 6.1. The results are shown in Table 6.2.1.

Dataset	Classification ratio
Original with Naive Discretization	76%
$\alpha = 0.90$	81%
$\alpha = 0.70$	72%
Best results (Entropy Discretization)	97%

Table 6.2.1 Summary of results

Observing that after applying the interval merging algorithm, we actually got an increase in the classification ratio, we can assume that a large number of intervals will give pretty specific rules, that have less validity for new objects, than a set of fewer intervals.

The interval merging algorithm can prove its usefulness for datasets where the training set is likely to have some degree of equal distributed noisy objects. By using a discretization algorithm that ensures no reduction of initial consistency level, and applying some interval

merging algorithm as a postprocessing of the discretization, we can have more detailed control over the process.

Chapter 7 Further work

The area of discretization is an area in which research in different fields can be applied. Several techniques from pattern recognition and vector quantization can be used in non-orthogonal discretization. This a large area of research with a lot of work to be done.

Several experiments can be carried out with different measures for entropy of a dataset, possibly improving the results of the Entropy-based discretization algorithm.

Research is being carried out within the area of non-orthogonal hyperplanes [2], and methods and algorithms from this research can also be applied to orthogonal discretization.

Another interesting field of discretization is to apply some discretization algorithm on the decision attribute.

For the algorithms presented in this report, their implementation can be optimized with regards to speed.

More testing could be performed to validate the usefulness of the proposed interval merging algorithm. Also, other heuristics for merging intervals could be applied. (Voting, reducing to a predefined number of cuts, etc.)

Appendix A Implementation

The Rosetta framework[15] is a general C++ class library for Rough Set data analysis. The Rosetta framework was developed by the Knowledge Systems group at the Institute of Computer and Information Science, The Norwegian University of Science and Technology.

Prior to the development of Rosetta, a set of C++ modules, called RSES, was developed by the University of Warsaw, Poland. The computational kernel of the RSES is encompassed in the Rosetta system as legacy code.

The Rosetta system is designed as two separate parts:

- The Rosetta kernel. A set of portable classes written in C++ with extensive use of templates and smart pointers.
- The Rosetta frontend. A user-friendly frontend to the Rosetta kernel. Currently only available for the 32-bit Microsoft Windows 95 and Microsoft Windows NT platform.

The design of the Rosetta kernel is based upon ideas found in [16].

For testing some of the algorithms described in the chapters above, implementation into the Rosetta framework was performed. The following algorithms was implemented in Rosetta as a part of this project.

- ChiMerge (class ChiSquareScaler)
- Entropy Discretization (class EntropyScaler)
- Simple Discretization (class SimpleScaler)
- Interval Merger (class IntervalMerger)

In addition, the Orthogonal Hyperplane algorithm for discretization was already implemented in Rosetta.

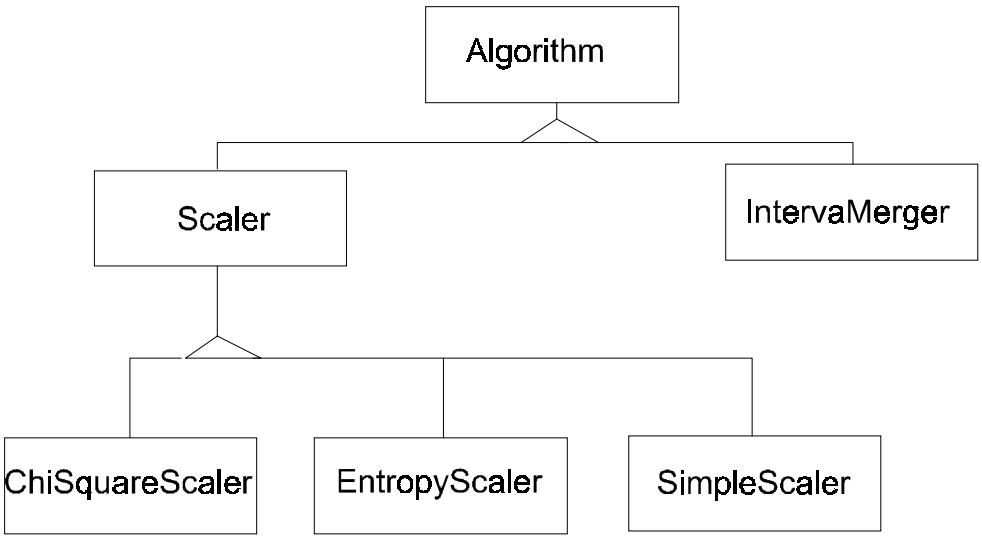


Figure 10 Partial Rosetta kernel class hierarchy with new algorithms

The classes for the algorithms implemented fit into the hierarchy as shown in Figure 10. The implementation extensively used the *Standard Template Library* [17], and the Handle-classes found in the Rosetta kernel. The Handle classes encapsulate pointers into smart pointers, and automates simple memory management.

Appendix B Bibliography

[1] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth (1996). *From Data Mining to Knowledge Discovery: An Overview.* In U.Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy. *Advances in Knowledge Discovery and Data Mining*. AAAI Press/MIT Press.

[2] N.H. Son, N.S. Hoa, A. Skowron (1995). *Searching for features defined by hyperplanes.* ICS Research report 69/95.

[3] T.W. Rauber, D. Coltuc, A. S. Steiger-Garcao (1993). *Multivariate Discretization of Continuous Attributes for Machine Learning.* In *Proceedings of the 7th International Symposium on Methodologies for Intelligent Systems.* Trondheim Norway, June 15-18, 1993.

[4] Z. Pawlak (1991). *Rough sets: Theoretical aspects of reasoning about data.* Dordrecht: Kluwer.

[5] T. Mollestad (1997). *A Rough Set Approach to Data Mining : Extracting a Logic of Default Rules from Data*. PhD Thesis. IDI NTNU.

[6] R. Kerber (1992). *ChiMerge : Discretization of numeric attributes.* In AAAI-92, *Proceedings Ninth National Conference on Artificial Intelligence*, pp. 123-128. AAAI Press/MIT Press.

[7] H. Liu, R. Setiono. *Discretization of Ordinal Attributes and Feature Selection.* In *Proceedings of the 7th International Conference on Tools with Artificial Intelligence*, Washington D.C., Nov 1995. pp. 388-391.

[8] U. Fayyad, K.B. Irani. *Multi-Interval discretization of continuos attributes as preprocessing for classification learning*. In *Proceedings of the 13th international Join Conference on Artificial Intelligence*, Morgan Kaufmann, pp. 1022-1027.

[9] J. Dougherty, R. Kohavi, M. Sahami (1995). *Supervised and Unsupervised Discretization of Continuous Features.* In *Machine Learning: Proceedings of the Twelfth International Conference*. Morgan Kaufmann Publishers.

[10] J.R. Quinlan (1995). *C4.5 Programs for Machine Learning.* Morgan Kaufmann Publishers.

[11] J.R. Quinlan (1996). *Improved Use of Continuous Attributes in C4.5*. In *Journal of Artificial Intelligence Research* 4, 1996 pp. 77-90.

[12] A. Skowron, N. H. Son (1995). *Quantization of Real value Attributes: Rough Set and Boolean reasoning approach.* In *Proceedings of the second International Joint Conference on Information Sciences*, Wrightswille Beach, NC, USA. Sept. 28 – Oct. 1. pp. 34-37.

[13] N.H. Son, N.S Hoa (1997). *Some efficient algorithms for Rough Set methods.*

[14] P.M. Murphy (1997), UCI Repository of Machine Learning Databases and Domain Theories. At http://www.ics.uci.edu/~mlearn/MLRepository.html.

[15] A. Øhrn, J. Komorowski (1997). *ROSETTA - A Rough Set Toolkit for Analysis of Data*. In *Proceedings of the 3rd International Joint Conference on Information Sciences*, Durham, NC, USA. Mar. 1-5. Vol. 3, pp. 403-407.

[16] E. Gamma, R. Helm, R. Johnson, J. Vlissides (1995), *Design Patterns - Elements of Reusable Object-Oriented Software.* Addison-Wesley Publishing Company, Inc.

[17] A. Stepanov, M. Lee (1995). *The Standard Template Library*. Available at http://www.cs.rpi.edu/~musser/doc.ps.