Department of Computer and Information Science

Norwegian University of Science and Technology

Fault Diagnosis in Rotating Machinery Using Rough Set Theory and ROSETTA

78070 DATABEHANDLING PROSJEKTARBEID

Torgeir Rhodén Hvidsten

April 30, 1999

Abstract

This report outlines an approach to use rough set theory and the ROSETTA system in the field of fault diagnosis in rotating machinery. The complexity of rotating machinery makes fault diagnosis a difficult task. Several computer based methods exist using mathematical modelling together with fuzzy logic, neural networks, faults matrices, machine specific experience and simulation. The ROSETTA system takes a machine learning approach to fault diagnosis by inducing rules on the basis of measured data classified by a domain expert.

Data used in this work was collected from 15 different measurement points on a large diesel engine. Six different machine states were recognised, five fault states and one normal state. Our primary goal was twofold. First, to establish whether machine specific knowledge could be used indirectly in order to reduce the amount of data one has to consider when inducing rules in ROSETTA. Second, to learn how much data is needed to induce effective rules in ROSETTA.

Our analysis shows that machine specific knowledge not only can be used to reduce the amount of data one has to consider, but also to obtain considerably better results. ROSETTA is proven to be a powerful tool for fault diagnosis, comfortably exceeding what domain experts consider to be good results.

Preface

This report was written in connection with the course *78070 Databehandling prosjektarbeid* held by the Department for Computer and Information Science (IDI), a faculty at the Norwegian University of Science and Technology (NTNU). The project is a result of the cooperation between the Knowledge System Group at IDI and the Department of Marine Machinery. Professor *Jan Komorowski* was my main supervisor through this project, while Professor *Maurice F. White* and PhD student *Bai Guanglai* was my contacts and domain experts on rotating machinery.

I have been working partly together with another student on this project whose name is Marte Skarstein Bjanger. The area we were investigating was the same, but we have been looking at two different machines, and consequently we have been working with different problems and data sets. Because of the similarity of our work, we wrote the first five chapters of the report, which presents the problem area and the theory needed, in collaboration.

Acknowledgement

I would like to thank Maurice F. White and Bai Guanglai for providing me with the data and for helping me to understand them. I would also like to thank Alexander Øhrn for always being available in his office, answering my many questions about rough set theory and ROSETTA. Finally, I would like to thank Marte S. Bjanger for our many eagering discussions, and Dyre Tjeldvoll for freeing me from the slavery of *Konvert.exe*.

Trondheim, April 30, 1999

Torgeir R. Hvidsten

Contents

1 Introduction **1**
- 1.1 Our Task . . . 1
- 1.2 Reader's Guide . . . 2

2 Data Mining and Knowledge Discovery **3**

3 Diagnosis of Rotating Machinery **7**
- 3.1 Problem Area . . . 7
- 3.2 Using Experts . . . 8
- 3.3 Using Mathematical Modelling . . . 8
- 3.4 Using Expert Systems . . . 9
  - 3.4.1 Fuzzy Logic . . . 9
  - 3.4.2 Inverse Modelling . . . 11
  - 3.4.3 Neural Networks . . . 11
  - 3.4.4 Fault Matrices . . . 12

4 Rough Sets **15**
- 4.1 Introduction . . . 15
- 4.2 Information Systems . . . 15
  - 4.2.1 Indiscernibility . . . 16
- 4.3 Decision Systems . . . 17
- 4.4 Set Approximation . . . 18
- 4.5 Reducts . . . 21
  - 4.5.1 Discernibility matrices . . . 21
  - 4.5.2 Discernibility functions . . . 22
- 4.6 Decision rules . . . 23

5 The Rough Set Method and ROSETTA **25**

5.1 Analysis Steps . . . 25
5.2 Preprocessing . . . 27
5.2.1 Completion . . . 27
5.2.2 Discretisation . . . 28
5.3 Reduct Computation . . . 29
5.4 Rule Generation . . . 31
5.5 Classification . . . 33
5.6 Evaluating the Classifier . . . 34

6 Problem Description **37**
6.1 The Machine . . . 38
6.2 The Data . . . 40
6.3 The Experiment . . . 41

7 Experiments and Results **45**
7.1 Preprocessing . . . 45
7.2 Using Objects from the Whole Range of Speeds . . . 46
7.3 Using Data from the Region Around the Optimal Operating Point . . . 48
7.4 Combining the two Methods . . . 48
7.5 Using Mean Value Objects . . . 49

8 Discussion/Conclusion **51**
8.1 Evaluation of the Results . . . 51
8.2 Comparison to other Methods . . . 55
8.3 Further Work . . . 55

A Preprocessing scripts **61**
A.1 konvert.prl . . . 61
A.2 make_table.prl . . . 62
A.3 operate30.prl . . . 65
A.4 duplicate.prl . . . 67
A.5 mean_duplicate.prl . . . 69

Chapter 1

Introduction

1.1 Our Task

Our work has been done in connection with cooperative work between two departments at NTNU, the Knowledge Systems group at the department of Computer and Information Science, and the department of Marine Machinery. The Knowledge Systems group is, among other things, concerned with different applications of methods based on rough set theory. The department of Marine Machinery has been interested in using several knowledge based methods and system models in the field of diagnosis in rotating machinery, among them the rough set method.

The work is carried out in connection with an EU research project on model based diagnosis of machinery in power stations. The research project is aimed at the development of methods for diagnosis of fault states and prognosis of the remaining lifetime of rotating machinery as gas turbines, steam turbines, generators and pumps.

The given assignment for our work was:

Model based diagnosis of machinery

The object of the work is to test methods for detection and evaluation of fault states. In particular, do tests with a diagnosis system based on the rough set method.

1.2 Reader's Guide

This report is roughly divided in two parts. The first part presents the background for our work, and gives the reader necessary knowledge on the theory upon which our work is based. The second part presents our work and a discussion on the results obtained. The reader is not expected to have any previous knowledge of rough set theory or diagnosis of rotating machinery.

The first part of the report starts with a general description of the field of Knowledge Discovery and Data Mining in chapter 2. This is to give the reader a general introduction to the area we are concerned with from a Computer Science point of view. Then in chapter 3, we give an overview of problems that arise in the field of diagnosis in rotating machinery, and what methods have been looked at to address these problems. Chapter 4 gives an introduction to rough set theory, and chapter 5 elaborates this theory by giving a description of ROSETTA, which is a system based on rough set methods.

In chapter 6 we present the specific problems addressed in this report. We also present the data we were provided with and explain how these were adapted to our analysis method. Chapter 7 explains the experiments in detail and presents the results of our work. Finally, a discussion on the obtained results is presented in chapter 8. In this chapter we also suggest further work to be done in the same field.

Chapter 2

Data Mining and Knowledge Discovery

Electronic measurement equipment and low cost storage media have the last few decades made us able to store large amounts of information. A large part of this information is primitive data collected in huge amounts, and a steadily increasing amount of this kind of data remains unanalysed. Consequently, the word *data* should not be mixed with the word *knowledge* which denotes the results of further analysis of data. The relation between data and knowledge is illustrated in Figure 2.1.

Knowledge Discovery (KD) is generally defined as the process of extracting nontrivial, previously unknown and potentially useful information from data. As defined in [1], this process is highly interactive and iterative, and includes the following steps:

1. The selection, cleaning, transformation and projection of data.
2. Mining the data to extract pattern and appropriate models.
3. Evaluation and interpretation of the extracted patterns.
4. Consolidation of the knowledge, resolving conflicts.
5. Making the knowledge available for use.

In this definition Data Mining (DM) is viewed as a step in the KD process. Thus, DM is reduced to the task of applying algorithms for extracting

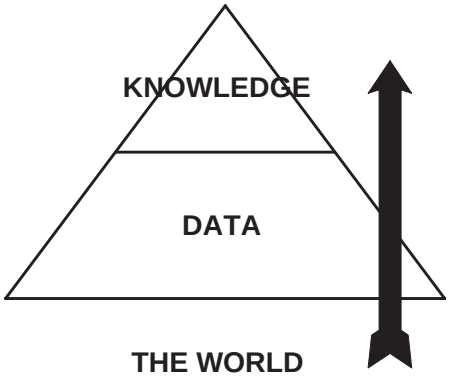

Figure 2.1: The information pyramid: Data are primitive pieces of information measured form the world, while knowledge is the information extracted from this data through analysis.

patterns or rules from the data.

KD can be viewed as a machine learning process. The input to this process is a set of data called the *training set*. This set can be viewed as a table where each row is an object and each column is an attribute describing a property of the given objects. This kind of table is referred to as an *information system*, and will be defined more formally in chapter 4. The machine learning approach assumes the presence of a domain *expert* who is able to classify the given objects in the information system by means of adding an attribute, the *decision attribute*, to the information system transforming it into a *decision system*. The actual learning in the KD process is now performed by means of extracting general patterns, often in terms of rules, from the decision system. Thus these patterns represent the knowledge initially held by the expert, but now also learned by the process.

A number of problems add to the uncertainty of the resulting knowledge extracted from a set of data. According to [1], databases typically have the following properties:

- They maintain *dynamic data*; the data kept in the DB may change over time.
- They may contain fields/attributes that are irrelevant to the learning process.

- They may have values missing for any object.
- They are *noisy*, i.e. they may contain errors or outliers.

The hypothesis behind any machine learning approach to trying to extract general knowledge is that the training set needs to be representative for the given problem domain. The four properties given above are clearly a threat to this assumption. Not taking that threat into account could either result in knowledge too specific to the given training set or, even worse, knowledge which is wrong relative to the given domain.

A more thorough introduction to the field of Knowledge Discovery and Data Mining can be found in [1].

Chapter 3

Diagnosis of Rotating Machinery

This chapter examines the field of fault diagnosis of rotating machinery. It looks at the challenges of using computer based systems as assistance, or even partially as substitution for human experts. It also describes some of the methods used for this purpose.

3.1 Problem Area

As stated earlier, the problem area of our work is diagnosing faults in rotating machinery. We are concerned with how it is possible to analyse data measured from a machine in order to determine if a fault state has occurred in the machine, and more preferably *which* fault state has occurred. In order to do this, we have to examine large sets of measured data, which provide us with information about the general state of the machine. The problem is how to handle these data sets, and to interpret them in the correct way.

As defined in [2], a machinery environment consists of five basic parts:

1. A machine - characterised by its complexity, time-dependency and speed of evolution.
2. Sensory and measurement system - characterised by the large number of

data collected, the uncertainty of this data and the possible overlapping and conflict between different observations.

3. Process control system - used for on-line process control and sometimes also for off-line decision support.

4. Interface - presents to the human experts the data coming from the sensory and measurement system, and the process control system.

5. Human experts - both operators, maintenance team and specialists.

We will now discuss what kind of role fault diagnosis can play in such an environment.

3.2 Using Experts

In the machinery environment outlined above, the experts play a very important role. However, these experts are often rare, expensive to use and certainly not available 24 hours a day. In addition, human experts often make mistakes as a consequence of both the psychological and the environmental factors ([2]). Some of these problems could obviously have been solved, or at least have become less severe, using computer-based assistance, and in particular computer-based fault diagnosis systems. Since both maintenance staff and specialists are first of all needed when something goes wrong, or to control that nothing will go wrong in the near future, an on-line fault diagnosis system would dramatically reduce the needs for these experts. Also, since a computer never rests, an expert system is guaranteed to detect the fault as early as possible. That is, if it is capable of detecting the fault at all. The question is now how to build such a system.

3.3 Using Mathematical Modelling

Mathematical models of rotating machinery consist of systems of non-homogeneous differential equations, where the general analytical solutions often are unknown ([3]). This is due to the non-linearity and the complex form of these equations. Thus, using mathematical modelling alone as the basis for expert

systems seems difficult. In the following we will explain some methods which have been used to build expert systems.

3.4 Using Expert Systems

Different approaches have been used for the purpose of computer-based fault diagnosis in rotating machinery. Due to the mentioned problems of using mathematical modelling alone, most approaches use a combination of mathematical modelling and various other methods. One method is to use inverse modelling together with fuzzy logic and neural networks. Another method is to use mathematical modelling together with both fault matrices, machine specific experience and computer simulation. Some of these topics are outlined here.

The rough set method together with the ROSETTA system is conceptually a method on the same level as the above mentioned methods. However, since this is the method used in this project, rough set theory and the ROSETTA system are more thoroughly examined later.

3.4.1 Fuzzy Logic

We divide our discussion of fuzzy logic into three parts; fuzzy set theory, fuzzy logic and fuzzy classifiers. For more information, refer to [4] and [3]

Fuzzy Set Theory

Fuzzy set theory is used to specify how well an object satisfies a vague description. A fuzzy set can thus be defined as the set of pairs

$$\{\langle t, p(t)\rangle | t \in U\} \tag{3.1}$$

where U is a set of terms called the universe and p is a fuzzy predicate such that p: $U \rightarrow [0, 1]$. A fuzzy predicate is thus not a relation p': $U \rightarrow \{true, false\}$ as in ordinary predicate calculus. It is also important to notice that fuzzy set theory is really not a method for uncertain reasoning, but

rather a method for specifying how strongly a given term, t, holds for a given predicate, $p(t)$.

Fuzzy Logic

Fuzzy logic evaluates logical sentences containing fuzzy predicates as a function of the truth values of its components. This evaluation should be performed in the following manner:

$$T(\mathcal{A} \wedge \mathcal{B}) = min(T(\mathcal{A}), T(\mathcal{B})) \tag{3.2}$$
$$T(\mathcal{A} \vee \mathcal{B}) = max(T(\mathcal{A}), T(\mathcal{B})) \tag{3.3}$$
$$T(\sim \mathcal{A}) = 1 - T(\mathcal{A}) \tag{3.4}$$

where T is the evaluated fuzzy truth and $\mathcal{A}$, $\mathcal{B}$ are arbitrary logical formulae.

Fuzzy Classifier

In our context, fuzzy set theory is used to classify objects according to given classes. Such a classification can be done defining a fuzzy set as a set of pairs:

$$\tilde{A} = \{\langle u, w_A(u)\rangle | u \in U\} \tag{3.5}$$

where U is a set of objects called the universe and w is a fuzzy predicate called the membership function such that w: $U \rightarrow [0, 1]$. The membership function $w_A(u)$ estimates in which degree the element u from the universe U belongs to the set $\tilde{A}$. Consequently the closer the value is to 1.0, the more strongly the element belongs to the set.

Given a well defined data set D, this set can be partitioned into regions or classes, where each class can be defined by a membership function. Thus the class and its corresponding membership function together form a fuzzy set. Given a set of membership functions, $\{w_1, w_2, ..., w_n\}$, each defining a class or a region of the data set, a given object x belongs to $class_i$ if:

$$\forall w_j((j \in \{1, .., n\} \wedge j \neq i) \rightarrow (w_i > w_j)) \tag{3.6}$$

3.4.2 Inverse Modelling

As mentioned, mathematical models in rotating machinery consist of systems of non-homogeneous differential equations, where the general analytical solutions often are unknown. In addition, these models are often strongly simplified and require data that are difficult to measure. Thus, experts try to look for models which take data that are easy to measure as input, and give data that are not so easy to measure as output. The easiest way to do this seems to be to invert the already existing models rather than preparing new models. This is because the existing models contain knowledge collected over a long time and which have been carefully validated through experiments ([3]).

Consider as an example the existing mathematical model M:

$$M : (x_1, ..., x_I, z_1, ..., z_K) \rightarrow (y_1, ..., y_J) \tag{3.7}$$

Let $x_1, ..., x_I$ and $y_1, ..., y_J$ be known parameters and let $z_1, ..., z_K$ be unknown parameters. The experts now look for an inverted model N which can calculate these unknown parameters:

$$N : (x_1, ..., x_I, y_1, ..., y_J) \rightarrow (z_1, ..., z_K) \tag{3.8}$$

It should be noted that the inverse model N does not exist in the general case. As an example, the mathematical model M could be mapping input parameters with different values into the same output value. How can this model be reversed? One way of handling this problem is to apply a fuzzy classifier as described earlier ([3]).

The most common way to look for the inverse model is to consider a neural network. A neural network can be viewed as a trainable black box. Examples are generated using the mathematical model M, and these examples are then used to train the network to obtain the model or mapping N. A more thorough examination of neural networks are outlined next.

3.4.3 Neural Networks

Neural networks is a method of representing functions using networks of simple arithmetic computing elements. These simple arithmetic computing

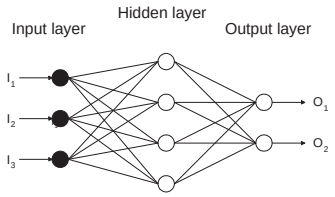

Figure 3.1: A 2-layer feed-forward neural network

elements correspond to the cells - called neurons - that perform information processing in the human brain. A neural network is thus a network of such neurons, or nodes, which are connected by links.

A numeric weight is associated with each node. These weights are the primary means of storage in neural networks. The learning in neural networks takes place by updating these weights. The external environment is "communicating" with the network trough some of the units, which are designated as input or output units. In order to bring the network's behaviour into line with the external environment, the weights of each node are modified. Each unit has input and output links. The units do a local computation based on the input from its neighbours. To train the neural network, the weights are initialised and then updated using a learning algorithm applied to a set of training examples for the task in question. For a thorough coverage of neural networks, see [5].

Figure 3.1 gives an example of a neural network. The particular neural network shown in the figure is a 2-layer feed-forward neural network. For a description of how this type of neural network can be used in diagnosis of rotating machinery, see [6].

3.4.4 Fault Matrices

Fault matrices are two-dimensional arrays. The rows represent objects, in our case different fault states. The columns represent attributes, in our case the symptoms that may occur in each fault state. In this way a fault matrix represents knowledge in a compressed way. From a fault matrix it is possible to form conjunctions of symptoms of a fault, and from these conjunctions we can conclude with either true or false for the fault in question. For a fault

to be detected, it is necessary that all characteristic symptoms of this fault are present.

According to [7], it is desirable that the fault matrix has the following properties:

1. Symptoms may take any truth-value, ranging from false over unknown to true.
2. The presence of a symptom may strengthen the belief in a hypothesis.
3. The absence of a symptom may strengthen the belief in a hypothesis.
4. The presence of a symptom may weaken the belief in a hypothesis.
5. The absence of a symptom may weaken the belief in a hypothesis.
6. The use of absolute values should be avoided. The relative change of a value should be used instead in order to facilitate tailoring of the system to particular machines.
7. It should be possible to assign prior beliefs to each possible fault reflecting the experienced distribution of faults for particular machines.
8. Under development of a new fault matrix it should be possible to apply different treatments of uncertainty using the same knowledge representation in order to find the one which is best suited for the problem.

Vibration analysis is a well-suited area for using fault matrices, since the inference chains usually are kept very short. In vibration diagnosis, the symptoms are usually directly connected to a fault, which is an advantage when fault matrix representation is used.

Two systems that have been developed based on fault matrices for use in diagnosis of rotating machinery are MATRIX and VIBEX. For a thorough description of these systems, see [7] and [8].

Chapter 4

Rough Sets

This chapter gives an introductory description of rough set theory. It is not meant to be a complete coverage of the theory, it merely gives an overview of the main concepts. The main concepts are throughout the chapter illustrated by an example. For a more thorough description of the theory, the reader is referred to [9].

4.1 Introduction

The process of dividing a universe of objects into different categories is called classification. Rough set theory deals with the analysis of this classificatory property of a set of objects. If you have large data sets, acquired from measurements or from human experts, these data sets may represent vague knowledge, for instance uncertain or incomplete knowledge. Rough set theory provides the means to discern and classify objects in data sets of this type, when it is not possible to divide the objects into defined categories.

4.2 Information Systems

In rough set theory, knowledge is represented in information systems. An information system is a data set represented in a table. Each row in the

table represents an object, for instance a case or an event. Each column in the table represents an attribute, for instance a variable, an observation or a property. To each object (row) there are assigned some attribute values.

An information system, $\mathcal{A}$, is defined as:

$$\mathcal{A} = (U, A) \tag{4.1}$$

U - non-empty finite set of objects called the universe
A - non-empty finite set of attributes such that

$$a : U \rightarrow V_a \text{ for every } a \in A$$

V_a - the value set of a

Example: To illustrate the concept of information system, we present an example. The example will be used throughout this chapter. The example information system is shown in Table 4.1.

The *objects* (rows) in this example are bottles of red wine. Different *attributes* (columns) are measured for each bottle. The measured attributes are *wine district*, *main grape variety*, *vintage* and *storage temperature*.

	Wine district	Main grape variety	Vintage	Storage temp.
x_1	Bordeaux	Cabernet Sauvignon	1992	12-15
x_2	Rhône	Syrah	1992	<12
x_3	Chile	Cabernet Sauvignon	1995	12-15
x_4	Bordeaux	Merlot	1995	>15
x_5	Chile	Cabernet Sauvignon	1995	12-15
x_6	Rhône	Merlot	1992	12-15
x_7	Bordeaux	Merlot	1995	>15
x_8	Chile	Merlot	1992	<12

Table 4.1: Example information system

■

4.2.1 Indiscernibility

One of the most important concepts of rough set theory is *indiscernibility*, which is used to define *equivalence classes* for the objects.

Given a subset of attributes $B \subseteq A$, each such subset defines an equivalence relation $IND_A(B)$ called an *indiscernibility relation*. This indiscernibility relation is defined as:

$$IND_A(B) = \{(x, x') \in U^2 | \forall a \in B, a(x) = a(x')\} \tag{4.2}$$

Equation 4.2 states that the subset of attributes, B, will define a partitioning of the universe into sets such that each object in a set cannot be distinguished from other objects in the set using only the attributes in B. The sets which the objects are divided into are called *equivalence classes*.

Example: In the information system shown in Table 4.1, we see that objects x_3 and x_5, x_4 and x_7 are (pairwise) *indiscernible*.

Example indiscernibility relations from the table:

$IND(WineDistrict) = \{\{x_1, x_4, x_7\}, \{x_2, x_6\}, \{x_3, x_5, x_8\}\}$

$IND(WineDistrict, MainGrapeVariety, Vintage, StorageTemp.) =$
$\{\{x_1\}, \{x_2\}, \{x_3, x_5\}, \{x_4, x_7\}, \{x_6\}, \{x_8\}\}$

■

4.3 Decision Systems

If a new attribute is added to the information system, and this attribute represents some classification of the objects, the system is called a *decision system*. We get:

$$\mathcal{A} = (U, A \cup \{d\}) \tag{4.3}$$

d - the *decision attribute*.

The elements of A are called *conditional attributes* or *conditions*.

The decision is not necessarily constant on the equivalence classes. That is, for two objects belonging to the same equivalence class, the values of the decision attribute may be different. In this case, the decision system is *inconsistent (non-deterministic)*. If a unique classification can be made for all the equivalence classes, the system is *consistent (deterministic)*.

Example: We add information to the information system in Table 4.1 by introducing a decision attribute. This additional attribute states whether or not the bottle of wine has been stored for a sufficient period of time. We now have a decision system, which is shown in Table 4.2.

	Wine district	Main grape variety	Vintage	Storage temp.	Decision
x_1	Bordeaux	Cabernet Sauvignon	1992	12-15	Drink now
x_2	Rhône	Syrah	1992	<12	Hold
x_3	Chile	Cabernet Sauvignon	1995	12-15	Drink now
x_4	Bordeaux	Merlot	1995	>15	Drink now
x_5	Chile	Cabernet Sauvignon	1995	12-15	Hold
x_6	Rhône	Merlot	1992	12-15	Hold
x_7	Bordeaux	Merlot	1995	>15	Drink now
x_8	Chile	Merlot	1992	<12	Hold

Table 4.2: Example decision system

■

From this example we see some of the problems that rough set theory addresses. For instance, objects x_3 and x_5 belong to the same equivalence class, but they are classified differently. They have the exact same values for the conditional attributes, but they have different values for the decision attribute. This means that the information system in this example is inconsistent. The question is then: How can we address the problem that arises due to the impossibility of distinguishing between the two objects using only the information given from the equivalence classes?

4.4 Set Approximation

In order to classify an object based only on the equivalence class in which it belongs, we need the concept of *set approximation*. Given an information system, $\mathcal{A} = (U, A)$, and a subset of attributes, $B \subseteq A$, we would like to approximate a set of objects, X, using only the information contained in B.

We define:

B-lower approximation of X:

$$\underline{B}X = \{x | [x]_B \subseteq X\} \tag{4.4}$$

B-upper approximation of X:

$$\overline{B}X = \{x | [x]_B \cap X \neq \emptyset\} \tag{4.5}$$

The lower approximation is the set containing all objects for which the equivalence class corresponding to the object is a subset of the set we would like to approximate. This set contains all objects which with certainty belong to the set X.

The upper approximation is the set containing the objects for which the intersection of the object's equivalence class and the set we would like to approximate is not the empty set. This set contains all objects which possibly belong to the set X.

We now define the *boundary region*:

B-boundary region of X:

$$BN_B(X) = \overline{B}X - \underline{B}X \tag{4.6}$$

This set contains the objects that can not be classified as definitely inside X nor definitely outside X. A set is *rough* if $BN_B(X) \neq \emptyset$.

The concepts of lower approximation, upper approximation and B-boundary region are illustrated in Figure 4.1.

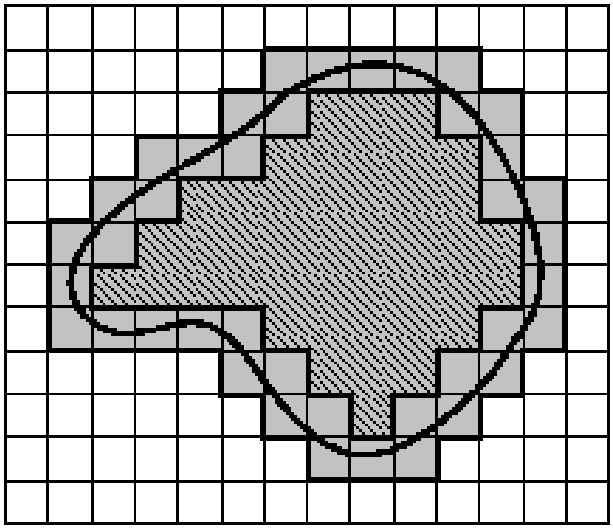

Figure 4.1: Set approximation

Example: Approximating the set of wine bottles that can be drunk now is shown in Figure 4.2.

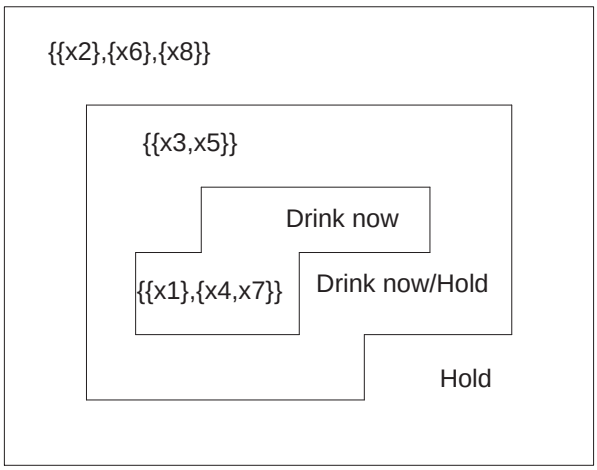

Figure 4.2: Approximation of the set **Drink now**

■

4.5 Reducts

Sometimes not all of the knowledge in an information system is necessary to divide the objects into classes. In these cases we can reduce the knowledge. Reducing the knowledge results in *reducts*.

A *reduct* is a minimal set of attributes, $B \subset A$, such that

$$IND_A(B) = IND_A(A) \tag{4.7}$$

A reduct is a combination of attributes that will make you able to discern between objects as well as you would if you used all attributes.

Example: To discern between the different equivalence classes in the example in Table 4.1, only the attributes *wine district* and *main grape variety* are necessary. Thus, $\{WineDistrict, MainGrapeVariety\}$ is an example of a reduct:

$$IND_A(\{WineDistrict, MainGrapeVariety\}) = IND_A(A)$$

■

Reducts can be computed on the basis of discernibility matrices and discernibility functions.

4.5.1 Discernibility matrices

A *discernibility matrix* of $\mathcal{A}$ is a symmetric $n \times n$ matrix with entries

$$c_{ij} = a \in A | a(x_i) \neq a(x_j) \text{ for } i, j = 1, ..., n.$$

The entries for each object are thus the attributes that are needed in order to discern object i from object j.

Example: The discernibility matrix for our example in Table 4.1 is shown in Table 4.3. For readability, the four attributes are abbreviated d, g, v, s.

We see that since objects x_3 and x_5 are indiscernible, the entry in the matrix here is the empty set. This means that there are no attributes that will discern between these two objects.

	$[x_1]$	$[x_2]$	$[x_3]$	$[x_4]$	$[x_5]$	$[x_6]$	$[x_7]$	$[x_8]$
$[x_1]$	$\emptyset$							
$[x_2]$	d, g, s	$\emptyset$						
$[x_3]$	d, v	d, g, v, s	$\emptyset$					
$[x_4]$	g, v, s	d, g, v, s	d, g, s	$\emptyset$				
$[x_5]$	d, v	d, g, v, s	$\emptyset$	d, g, s	$\emptyset$			
$[x_6]$	d, g	g, s	d, g, v	d, v, s	d, g, v	$\emptyset$		
$[x_7]$	g, v, s	d, g, v, s	d, g, s	$\emptyset$	d, g, s	d, v, s	$\emptyset$	
$[x_8]$	d, g, s	d, g	g, v, s	d, v, s	g, v, s	d, s	d, v, s	$\emptyset$

Table 4.3: Discernibility matrix.

■

4.5.2 Discernibility functions

From the discernibility matrix, we can build a *discernibility function*. A *discernibility function* f_A for an information system $\mathcal{A}$ is a Boolean function of m Boolean variables $a_1^*, ..., a_m^*$ (corresponding to the attributes $a_1, ..., a_m$) defined as below, where $c_{ij}^* = \{a^* | \ a \in c_{ij}\}$

$$f_A(a_1^*, ..., a_m^*) = \wedge\{\vee c_{ij}^* | 1 \leq j \leq i \leq n, c_{ij} \neq \emptyset\}$$

The discernibility function is a conjunction of all the entries in the discernibility matrix that are not the empty set. The conjunction may, if possible, be simplified. The results of simplification are the possible reducts for the information system.

It is also possible to generate a discernibility function from the discernibility matrix for one of the objects in the information system. This is done by looking at only one row (or column) in the discernibility matrix, and form a conjunction of all the entries in this row (or column). When we simplify this conjunction, we get possible reducts for the particular object in question.

4.6 Decision rules

For decision systems, $\mathcal{A} = (U, A \cup \{d\})$, we would like to find an approximation of the decision, d. This can be done by constructing *the decision-relative discernibility matrix* of $\mathcal{A}$. This matrix tells us how to discern an object from objects belonging to another decision class. The process of computing this matrix is called computing the discernibility matrix modulo the decision attribute.

If $M(A) = (c_{ij})$ is the discernibility matrix of a decision system $\mathcal{A}$, the decision-relative discernibility matrix of $\mathcal{A}$ is defined as:

$M^d(A) = (c^d_{ij})$ assuming $c^d_{ij} = \emptyset$ if $d(x_i) = d(x_j)$ and $c^d_{ij} = c_{ij} - d$, otherwise.

From the reducts computed from this discernibility matrix, we can generate decision rules for classification of the objects.

Example: A reduct for the decision system in Table 4.2 is $\{WineDistrict, MainGrapeVariety\}$. This reduct will, for object x_2, produce the decision rule:

Wine District = Rhône $\wedge$ *Main Grape Variety* = Syrah $\Rightarrow$ *Decision* = Hold

■

Chapter 5

The Rough Set Method and ROSETTA

In this chapter we will describe the rough set method in more detail. In particular, we will look at how the ROSETTA system implements this method.

The ROSETTA system is a toolkit for analysis of tabular data using rough set theory. It consists of a Windows NT interface against a kernel containing various algorithms for preprocessing, reduct computation, rule generation, rule filtering and classification.

For a more thorough description of the rough set method and the ROSETTA system, see [9] and [10]

5.1 Analysis Steps

Figure 5.1 shows the analysis steps when using the ROSETTA system. ROSETTA takes a machine learning approach to the classification task. The decision table is imported into ROSETTA and divided into a *training set* and a *test set*. The idea behind the method is now to use the training set to induce rules, and then use the test set to validate these rules. In general, non-categorical attributes need to be discretised in order to induce effective rules. The cut-off points generated during the discretisation step are produced using the training set. The test set is then discretised using these cuts. Rules are

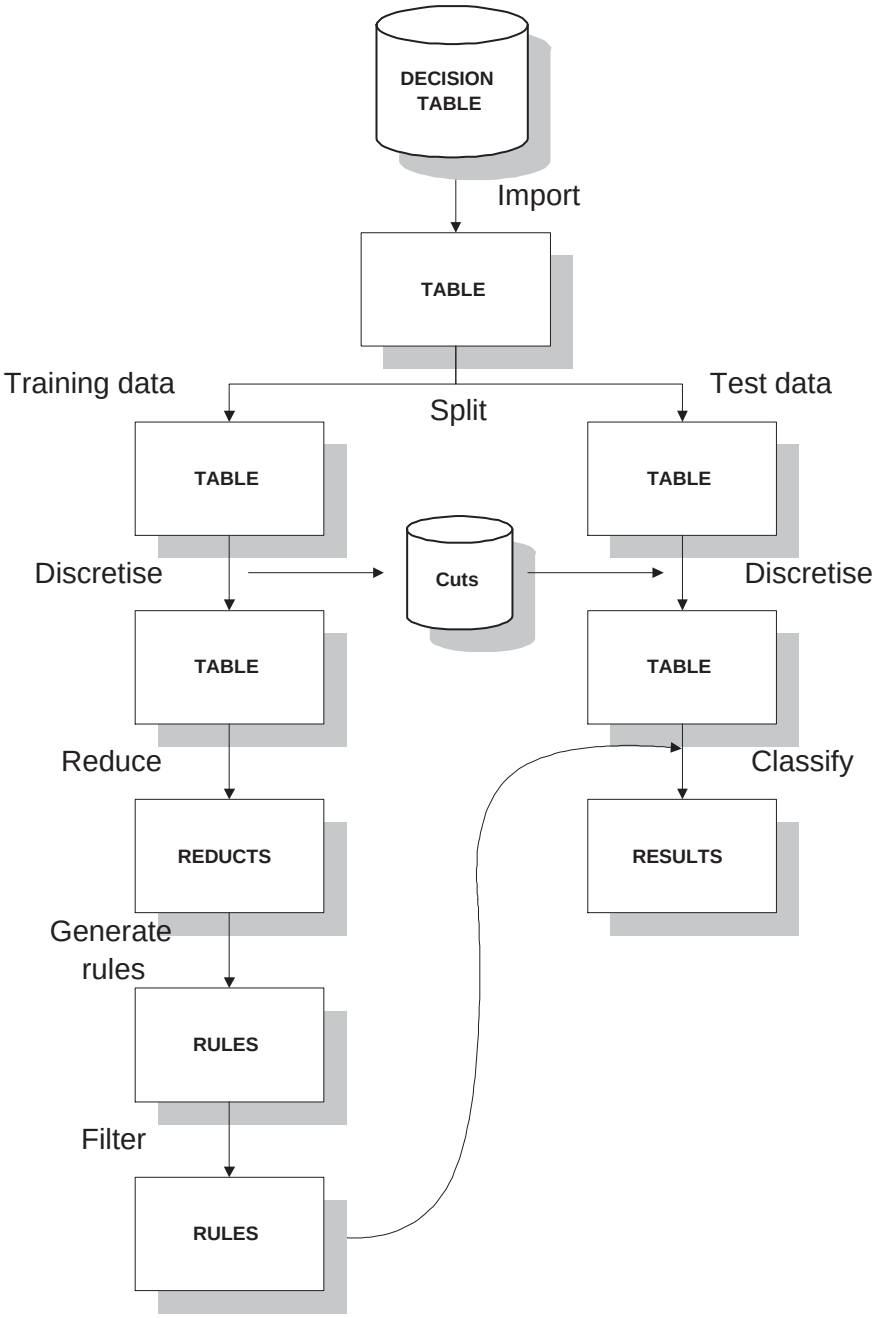

Figure 5.1: How to use the rough set method and the ROSETTA system to induce and validate a classifier

then generated on the basis of the discrete training table using the computed reducts. The results from applying the generated rules on the discrete test set, indicate how successfully the rules classified the objects in the test set.

This method can be carried out for two purposes. First, the rule set can be viewed as a classifier, used for the purpose of classifying only. Second, the computed reducts and the generated rules can be used by domain experts to learn more about the data. The last approach often requires a small set of rules for the human expert to examine, thus rule filtering can be carried out.

We will now examine the steps briefly described here in more detail.

5.2 Preprocessing

Before the data can be imported into ROSETTA they need to be collected and classified by a domain expert. Then they need to be transformed into a decision table. This involves quite a lot of work, since it often is not obvious how to interpret the data. Also, this is often an iterative process; as you learn more about the data through using the pipeline shown in Figure 5.1, you may want to view the data in a different way.

After the data has been imported into ROSETTA, two important preprocessing steps need to be carried out; completion and discretisation.

5.2.1 Completion

During measurements values often get lost. Objects with missing values are very undesirable, since ROSETTA might make rules on the basis of these missing values. Three algorithms exist in ROSETTA that take different approaches to the task of filling in these missing values:

1. Remove objects with missing values - this is often undesirable when very few objects are available or when the object in question represents a very rare event.
2. Fill in the missing values with the mean value of all non-missing entries for that attribute - alternatively, the computation of the mean values could be conditioned to the decision classes.

3. Expand every object with a missing value into multiple objects; one object for each possible value - alternatively, the set of possible values could be conditioned to the decision classes.

5.2.2 Discretisation

In ROSETTA, discretisation is reduced to searching for "cuts" that determine intervals. All values that lie within this interval is then mapped to the same value. Discretisation is necessary to ensure that the rules ROSETTA induces are not too specific. Some of the methods used for discretisation in ROSETTA are briefly explained here:

1. **Naive algorithm:** For each attribute:

   a) Order the objects according to the value.

   b) Scan the sorted objects and add a cut for that attribute midway between to neighbouring objects if the decision values for those objects differ.

   There also exists a semi-naive algorithm that selects a subset of the cuts found by the naive algorithm.

2. **Equal frequency binning:** For each attribute the algorithm discretises the given attribute into a number of intervals such that each interval contains approximately the same number of objects.

3. **Boolean reasoning:** This algorithm reduces the search for appropriate cut-off points to finding minimal Boolean expressions.

4. **Manual cuts:** Gives the user the opportunity to enter cuts manually.

It is reasonable to expect that the naive, and even the semi-naive, algorithm provides more cuts than is actually necessary because they only consider one attribute at a time. The boolean reasoning algorithm sees patterns in multiple dimensions by considering more than one attribute at a time. Thus this algorithm produces fewer cuts. It is often desirable to first use boolean reasoning and then use one of the other algorithms on the remaining attributes. Manual cuts are often provided by a domain expert, and can therefore be of great help in many situations.

5.3 Reduct Computation

Computing reducts is the task of finding minimal attribute subsets as explained in chapter 4. Note that ROSETTA only compute approximations to reducts. Finding a minimal reduct is NP-hard, but there exists good heuristics that compute sufficiently many reducts in an acceptable time. The theoretical maximum numbers of reducts one can compute from an information system $\mathcal{A} = (U, A)$ equals a binomial coefficient such that:

$$|RED(\mathcal{A})| \leq \binom{|A|}{\lceil |A|/2 \rceil} \tag{5.1}$$

To understand how reducts are computed in ROSETTA, we should consider the following definitions:

i) A *term* is a conjunction of literals.

ii) An *implicant* of a Boolean function f is a term p such that $p \leq f$.

iii) A *prime implicant* is an implicant of f that ceases to be so if any of its literals are removed.

iv) Given any information system $\mathcal{A} = (U, A)$: $B \in RED(\mathcal{A})$ if and only if $term(B)$ is a prime implicant of the discernibility function f_A.

Thus finding reducts boils down to finding prime implicants of the discernibility function. There exist many algorithms for reduct computation in ROSETTA. Some of them are explained here:

1. **The Johnson algorithm:** This algorithm invokes a simple greedy algorithm that computes a single reduct only. Consider an information system $\mathcal{A} = (U, A)$ where:

   i) f_A is the discernibility function (product of sums) constructed on the basis of the discernibility matrix as explained in chapter 4.

   ii) $w(s)$ denotes a weight for a sum s in f_A.

   A reduct B can now be found in two steps:

   a) Initialise B to the empty set.

b) While the function f_A has any sums left:
Add attribute a to B if a maximises $\sum w(s)$, where s occurs in f_A and a occurs in s. Delete all sums from f_A that contains a.

Running the Johnson algorithm results in a small set of reducts, and thus also a small set of rules, compared to the other algorithms available in ROSETTA.

2. **The genetic algorithm:** This is an implementation of a genetic algorithm for computing minimal hitting sets. Again consider an information system $\mathcal{A} = (U, A)$ and interpret its discernibility function as a set of sets in the following manner:

$$\mathcal{S}_{f_A} = \{M_A(x, y) | M_A(x, y) \text{ is used to construct } f_A\} \tag{5.2}$$

where $M_A(x, y)$ is the discernibility matrix and f_A is the discernibility function. A *hitting set* of $\mathcal{S}_{f_A}$ is now a set $A_h \subseteq A$ such that the intersection between A_h and every set in S is non-empty. A *minimal hitting set* is consequently a hitting set A_h that ceases to be a hitting set if any of its elements are removed. It should now be obvious that a hitting set of $\mathcal{S}_{f_A}$ is an implicant of f_A. And consequently a minimal hitting set of $\mathcal{S}_{f_A}$ is a prime implicant of f_A, or, as stated earlier, a reduct.

3. **Dynamic reducts:** Real world data almost always contain noise and other peculiarities. Only one single noisy object can change the discernibility matrix, thus one would like to find reducts that reveal the underlying, general pattern in the data set.

   Computing dynamic reducts from an information system $\mathcal{A} = (U, A)$ can be done in the following manner:

   a) Randomly sample a family of subsystems $\mathcal{S} = \{\mathcal{A}_i\}$ from $\mathcal{A} = (U, A)$ such that $\mathcal{A}_i = (U_i, A)$ and $U_i \subseteq U$.

   b) From each subsystem $\mathcal{A}_i \in \mathcal{S}$, compute RED($\mathcal{A}_i$).

   c) The reducts that occur the most often in step b) are now believed to be the most "stable".

   The actual reduct computation performed in b) can be done using one of the other two algorithms described. Dynamic reduct computation

are thus just a combination of normal reduct computation and resampling techniques.

In ROSETTA, reducts can be computed along two dimensions:

1. Reducts can be computed on the basis of both the discernibility matrix modulo the decision attribute and the discernibility matrix not modulo the decision attribute.

2. Reducts can be computed with:

   a) full discernibility: The discernibility function is built using all the entries in the discernibility function.

   b) object related discernibility: One discernibility function is built for each equivalence class using one row in the discernibility function.

When ROSETTA is used for classification purposes, reducts are computed modulo the decision attribute and with object related discernibility. However, full discernibility is often used when one is interested in general patterns rather than rules.

5.4 Rule Generation

Rules are generated on the basis of the computed reducts, and constitute one of the most important results of rough set data analysis.

Given a decision system $\mathcal{A} = (U, A \cup \{d\})$, a *descriptor* is defined as a simple expression $a = v$ where $a \in A$ and $v \in V_a$. A decision rule is now denoted $\alpha \rightarrow \beta$, where α is a conjunction of descriptors, formed by overlaying a reduct $B \in RED(\mathcal{A})$ over an object $x \in U$ and reading off the value of x for every $a \in B$, and β is the corresponding descriptor $d = d(x)$. α is called the rule's *antecedent* and β is called the rule's *consequent*. In ROSETTA rules are generated in this manner for every object and its related reducts. New occurrences of an already existing rule are discarded. Rules with equivalent antecedent and different decisions are merged. Thus, for an inconsistent decision system the consequent of a decision rule $\alpha \rightarrow \beta$ could be a disjunction of descriptors.

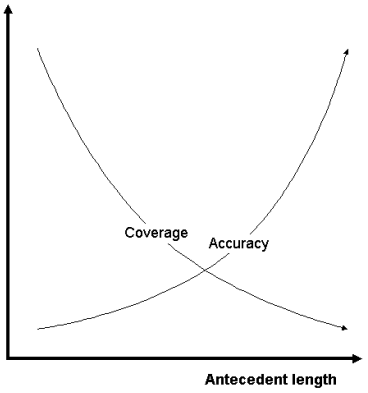

Figure 5.2: The relation between coverage and accuracy

There exist several units of measure for the quality of a given decision rule $\alpha \rightarrow \beta$:

1. **Support:** The number of rules that possess both property α and β.
2. **Accuracy:** Measures how trustworthy the rule is in drawing conclusion β on the basis of evidence α:

$$accuracy(\alpha \rightarrow \beta) = \frac{support(\alpha \cdot \beta)}{support(\alpha)} \tag{5.3}$$

   Equation (5.3) is often used as an estimate for the conditional probability $\Pr(\beta|\alpha)$.

3. **Coverage:** Measures how well the evidence α describes the decision class defined through β:

$$coverage(\alpha \rightarrow \beta) = \frac{support(\alpha \cdot \beta)}{support(\beta)} \tag{5.4}$$

   Equation (5.4) is often used as an estimate for the conditional probability $\Pr(\alpha|\beta)$.

When inducing rules one would like to obtain decision rules that are both accurate and have a high degree of coverage. Figure 5.2 shows graphically that as the antecedent grows long, the coverage decreases while the accuracy increases. Thus one would have to balance the trade-off between these two measures.

5.5 Classification

The set of rules induced on the basis of the computed reducts are often used to classify new and unseen objects. In this context the rule set is called a classifier. Given the classifier RUL and a test set, there exist different approaches to perform the actual classification. The most usual one is called *voting* and is described bellow.

Given object x to classify, voting is performed in the following way:

a) The set RUL is scanned for rules that have an antecedent that matches x. These rules are denoted $RUL(x)$ and are said to *fire* for the given object x.

b) If $RUL(x) = \emptyset$ the classification is undefined.

c) The different decisions indicated by the rules in $RUL(x)$ are given votes in an election process:

   i) Each rule $r \in RUL(x)$ are given a number of votes $votes(r)$ in favour of the decision class indicated by the rule. $votes(r)$ is typically based on the support of the rule r.

   ii) A normalisation factor $norm(x)$ is computed. $norm(x)$ is typically the number of votes given altogether.

   iii) A certainty coefficient that indicates the certainty of x belonging to the decision class β is then defined:

$$certainty(x, \beta) = \frac{votes(\beta)}{norm(x)} \tag{5.5}$$

The set of certainty coefficients given by the voting algorithm can now be interpreted in two different ways:

1. Choose the decision class with the highest certainty coefficient.
2. Prioritise a particular decision class by always choosing this particular decision class if it obtained a certainty coefficient above a given threshold. Else, choose the the decision class with the highest certainty coefficient.

How to evaluate the resulting classification is described next.

5.6 Evaluating the Classifier

In the following we will view a classifier **c** as a function that takes a given object x as input and gives a classification $\hat{d}_{\mathbf{c}}$ as output. The true classification of x will be denoted d(x).

One way of evaluating the result of applying a classifier **c** to a set of test objects, is to consider a *confusion matrix* C. The entry $C(i, j)$ is the number of objects that really belong to class i, but were classified by **c** as belonging to class j. Given the case where **c** is a binary classifier, i.e. where $V_d = \{0, 1\}$, the following would be a confusion matrix:

		$\hat{d}_{\mathbf{c}}$	
		0	1
d	0	TN	FP
	1	FN	TP

where TN is interpreted as "true negatives", TP as "true positives", FN as "false negatives" and FP as "false positives". Three important quantities in this context is *sensitivity*, *specificity* and *accuracy*:

$TP/(TP + FN)$	Sensitivity	$\Pr(\hat{d}_{\mathbf{c}} = 1 \mid d(x) = 1)$
$TN/(TN + FP)$	Specificity	$\Pr(\hat{d}_{\mathbf{c}} = 0 \mid d(x) = 0)$
$(TP + TN)/(TP + TN + FP + FN)$	Accuracy	

By further considering a binary classifier, with decision classes X_0 and X_1, we can define the relation $\hat{d}_{\mathbf{c}}$ as follows:

$$\hat{d}_{\mathbf{c}} : U \xrightarrow{\Phi} [0, 1] \xrightarrow{\theta} \{0, 1\} \tag{5.6}$$

Considering a given object x to classify, we now have the following:

$$\Phi(x) = certainty(x, X_1) \tag{5.7}$$

$$\theta(\Phi(x)) = \begin{cases} 1 & \text{if } \Phi(x) \geq \tau \\ 0 & \text{otherwise} \end{cases} \tag{5.8}$$

Here, τ is the threshold mentioned at the end of section 5.5. Remember that *certainty* was the output of the voting algorithm also explained in section 5.5.

The binary classifier can now be evaluated along two dimensions:

1. Discrimination: Measure of how good the classifier is at guessing the correct value for $d(x)$ when presented with object x.
2. Calibration: Measure of how close the output $\Phi(x)$ of a classifier is to the probability $\Pr(d(x) = 1|x)$.

Discrimination is an intuitive measure because it indicates how good a classifier is at classifying. Calibration is also very important, because human experts often want to use this value as a decision base rather than using the final classification given by θ.

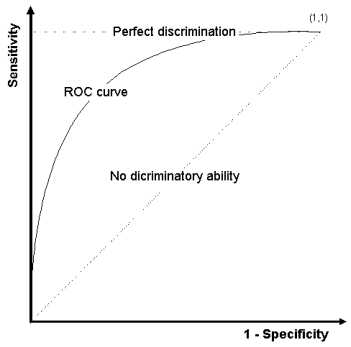

Figure 5.3: An example of an ROC curve

A frequently used graphical representation of discrimination is the *receiver operating characteristic* (ROC) curve. Equation 5.8 shows us that the output of a classifier depends on a selected threshold τ. An ROC curve describes the behaviour of a classifier as the threshold τ is varied across the full spectrum of possible values. Each point on the ROC curve represents a different 2×2 confusion matrix with different sensitivity and specificity, all defined by the given threshold value. An example of an ROC curve is given in Figure 5.3. It is common to interpret the *area under the ROC curve*, AUC, as the highest obtainable accuracy, given that the best threshold τ is used.

There exist other measures not discussed here, including calibration plot and various methods using statistical hypothesis testing.

Chapter 6

Problem Description

We now turn our attention to the specific problems addressed in this project. These problems are both specific to fault diagnosis and to the ROSETTA system. The data used was collected from a large diesel engine and classified by experts on rotating machinery. ROSETTA was then used to classify these data in order to answer the following main questions:

1. How many objects from each decision class are needed in order to classify objects from this machine?

2. Is it possible to reduce the amount of data one would have to consider by using expert knowledge about the given machine?

In addition, we were also interested in the answers of more general questions like; How can ROSETTA be used efficiently to solve practical problems specific to the given data? Which algorithms for preprocessing and reduct computation gives the best results for the given data?

In the following we will describe the machine and the collected data more thoroughly. We will also explain how we plan to carry out the experiments in order to answer the given questions.

6.1 The Machine

Figure 6.1 shows a picture of the large diesel engine used in this project and Table 6.1 shows some of its most important characteristics.

Figure 6.1: The large diesel engine considered in this rapport

Type:	Scania DS11
Diameter of cylinder:	127 mm
Length of stroke:	145 mm
Number of cylinders:	6
Number of tacts:	4
Rate of compression:	15:1 (nominal)
Max. nominal capacity:	186 kW (2200 rpm)
Max. load:	1024 (1050 rpm)
Turbo charger:	Holset type 4-550-224
Fuel system:	Direct injection (Bosch)
Opening pressure:	210kp/cm^2

Table 6.1: Selected data for the given machine

Two parameters which are important when collecting data from the machine are speed and load. Speed is measured in revolutions per minute (rpm) and indicates how fast the machine's rotor spins. Load is measured in newton meter (Nm) and indicates what kind of load the machine can handle at a given speed. Thus load is a function of speed. It seems reasonable to expect that load increases with increasing speed. However, this is only partially

correct. At a certain point the machine reaches its maximum capacity, and the load starts decreasing. This point is called the *optimal operating point*. Given the optimal operating point and its corresponding speed s, one can define a region below and above this speed given by a radius corresponding to 30% of s. In this region experts assume linearity. The relation between speed and load for the machine in normal condition is depicted in Figure 6.2.

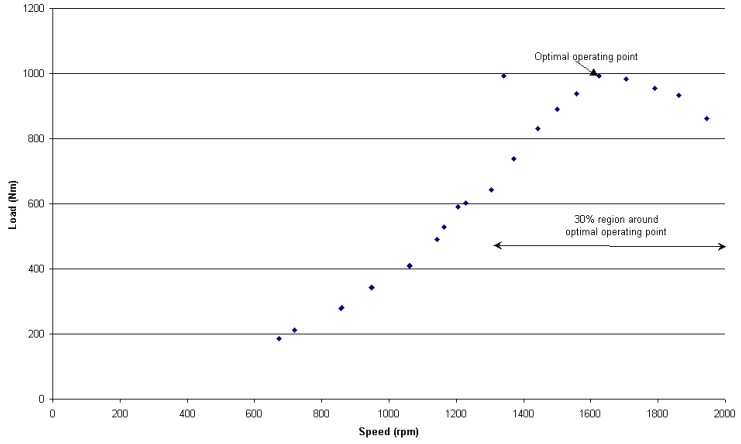

Figure 6.2: The relation between speed and load for the machine in normal condition. The optimal operating point and its corresponding 30% region is also indicated.

The diesel engine is known to be operating in one of six different states, five fault states and one normal state:

1. Normal.
2. Insufficient air.
3. Insufficient air & misfiring in cylinder 4.
4. Insufficient cooling.
5. Insufficient cooling & misfiring in cylinder 4.
6. Misfiring in cylinder 4.

Obviously one would like the machine to operate in the normal state. A fault should be detected as early as possible in order not to damage the machine. However, the machine can operate normally for years before a fault occur.

By way of comparison, a fault can develop in just a few hours. Hence, the machine should be under constant supervision in order to avoid the machine to enter one of the fault states.
The relations between the different states can be viewed as a state chart depicted in Figure 6.3.

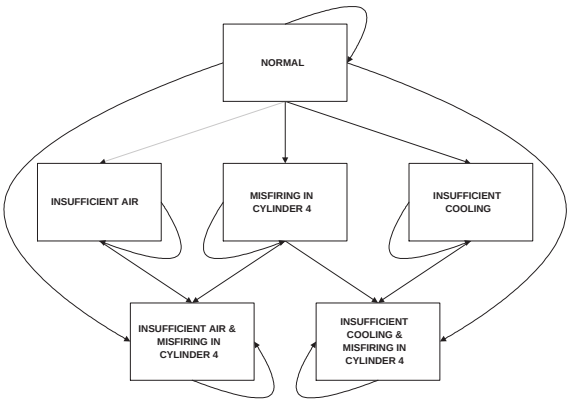

Figure 6.3: State chart showing the relations between the different machine states.

6.2 The Data

The diesel engine is equipped with a computer based measurement system which can measure a large number of parameters. In addition, some parameters need to be measured by hand. The six different states from Figure 6.3 was simulated, and measurements were done for increasing speed in the same manner as shown in Figure 6.2. Here each point, given a fixed speed and load, corresponds to an object in an information system as defined in chapter 4. Each object has 15 attributes, including speed and load. These attributes are shown in Table 6.2. In addition, each object belongs to a decision class. A decision class corresponds to the set of objects measured from one of the given machine states, thus there are six different decision classes. Consequently, the measured data constitute a decision system with 15 conditional attributes and one decision attribute.

The amount of data we have available in this project is not equally distributed

speed (rpm)	exhaust from turbo (2)
load (Nm)	charge air in compressor
charge air pressure (bar)	air temp. in the motor cell
exhaust pressure	temp. of cooling water from motor
consumption (cm^3/m)	exhaust temp. in turbo
turbo charger (rpm $\times$ 1000)	fuel temp.
cooling water in the motor	power (kW)
exhaust from turbo (1)	

Table 6.2: Attributes measured from the diesel engine.

over the different decision classes. The following objects are available: 25 objects from the *normal* decision class, 50 from *insufficient air*, 100 from *insufficient air & misfiring in cylinder 4*, 50 from *insufficient cooling*, 50 from *insufficient cooling & misfiring in cylinder 4* and 100 from *misfiring in cylinder 4*

6.3 The Experiment

We will now describe how we intend to carry out the experiments in order to answer the questions outlined in the beginning of this chapter.

When using accuracy as a measure of quality for a given classifier, one should be aware of two problems. First, accuracy can give a wrong impression if the objects are unequally distributed over the given decision classes. As an example, consider a decision system with two decision classes, X_0 and X_1, X_0 with 90 objects and X_1 with 10 objects. The objects in this decision system can now easily be classified with an accuracy of 0.90 simply by assigning decision class X_0 to every object. The obtained accuracy is now good, but the classifier is absolutely useless! The second problem with accuracy should be seen in relation with how ROSETTA induces rules and how ROSETTA classifies on the basis of these rules (described in chapter 5). A large number of objects from one decision class, would in turn give a large number of rules indicating this decision. Thus there is a large possibility that objects not belonging to this decision class will match some of the rules indicating this decision. Consequently, if enough votes are given to the wrong decision,

the objects will be classified incorrectly. Normally, the distribution of the amount of objects from each decision class should as far as possible reflect the distribution from the real world. However, in our data set the *normal* decision class would totally overshadow the *fault* decision classes, since a machine can spend years operating normally before a fault occurs. We will therefore have to use a larger number of objects from the *fault* decision classes compared with the real world, in order to be able to classify them at all. This can in some sense be justified noting the fact that using more objects from the fault decision classes only makes it easier for the voting process to detect these objects, thus giving a better classification altogether.
One should notice that using AUC in high degree evade these difficulties. Unfortunately, AUC can only be used in relation with binary classifiers. We will come back to this problem later.

For the purpose of this project we will use an equal amount of data from each decision class in the training sets. In this way we make comparison easy and we prioritise all decision classes equally. We will subsequently compare classifiers based on training sets with different amount of data; five objects from each decision class, ten objects from each decision class, ... , 50 objects from each decision class, with respect to accuracy. When too few objects are available from a given decision class, randomly selected objects from this class will be copied until enough objects are obtained. In the same manner objects will be discarded randomly from the decision classes in question when too many objects are available. Note, however, that the copying and discarding of objects only will be performed in the training set, never in the test set. Thus this has to be done after the decision table has been divided into a training set and a test set.
We will also try out a different approach when too few objects from a given decision class are available. This approach creates a new object generated on the basis of all the other objects in this decision class by computing the mean value of each attribute and then use these values as attribute values. Then this object will be copied until enough objects are obtained.

As mentioned, experts assume linearity in the region around the optimal operating point. This indicates that this region is stable and easy to model mathematically compared with other regions. Thus one would expect that using training and test sets only from this region would give better results than using objects from the whole range of speeds. Also, it would be interesting to use objects from the region around the optimal operating point as

training data and objects from the whole range of speeds as test data.

The two approaches outlined above, using different amounts of data for rule induction and using different ranges of speeds as the basis for rule induction, will be merged into one experiment. Also, different algorithms for reduct computation will be used and compared. The results from these experiments will be outlined next.

Chapter 7

Experiments and Results

In this chapter we will present the results from the experiments briefly explained in section 6.3. We will also explain the experiments themselves more carefully. Next, in chapter 8, we will try to interpret these results and possibly draw some conclusions.

Two main experiments have been carried out. One that used objects from the whole range of speeds in both the training and the test set, and one that used objects from the area around the optimal operating point in both the training and the test set. In each experiment rule sets, or classifiers, were induced on the basis of both 5, 10, 20, 30, 40 and 50 objects from each decision class. In addition, three different algorithms for reduct computation where used; Johnson algorithm, Genetic algorithm and dynamic reducts with Johnson algorithm.

7.1 Preprocessing

The computer based measurement system connected to the diesel engine outputs each object in a separate plain text file together with a lot of other data not relevant for these experiments. The Perl scripts written to extract the relevant data and make a decision table out of it can be seen in appendix A1 and A2.

Two algorithms for discretisation were used subsequently:

a) Boolean reasoning.

b) Equal frequency binning.

In other words, Equal frequency binning was used on the attributes which were not discretised by Boolean reasoning. The choice of algorithms was based on both examinations of the cuts generated and on the resulting accuracy obtained by using different algorithms for discretisation and reduct computation.

7.2 Using Objects from the Whole Range of Speeds

Experiments using objects from the whole range of speeds both in the training set and in the test set were carried out in the following manner:

1. The original decision table was imported into ROSETTA and then divided into a training set and a test set (90% - 10%).
2. The training set was exported out of ROSETTA and a Perl script was used in order to obtain the right distribution of objects according to the given experiment. That is, random selected objects were copied when too few objects were available or discarded when too many objects were available. The Perl script used for this purpose can be seen in appendix A4.
3. The training set, now containing an equal amount of objects from each decision class, was imported into ROSETTA once more.
4. Boolean reasoning and Equal frequency binning were used to discretise the training set and the resulting cuts were used on the test set.
5. Reducts were computed and rules were generated on the basis of the training set using Johnson algorithm, Genetic algorithm and dynamic reducts with Johnson algorithm.
6. The induced rule sets were used to classify the objects in the test set.
7. Step 2 - 6 were repeated with 5, 10, 20, 30, 40 and 50 objects from each decision class in the training set.

8. Step 1 - 7 were repeated four times, each time using a different split.

The results from these experiments can be viewed in Table 7.1. The RNG seed shown in the table is a number used by the random number generator in ROSETTA to randomly split the data set into a training set and a test set.

	Number of objects from each decision class	Measure of accuracy				
		Split 1	Split 2	Split 3	Split 4	Mean value
Johnson alg.	5	0.26	0.29	0.45	0.37	0.34
	10	0.45	0.26	0.45	0.24	0.35
	20	0.45	0.55	0.0.50	0.53	0.51
	30	0.66	0.82	0.71	0.68	0.72
	40	0.50	0.58	0.47	0.79	0.59
	50	0.61	0.63	0.74	0.74	0.68
Genetic alg.	5	0.37	0.45	0.58	0.37	0.44
	10	0.42	0.34	0.60	0.61	0.49
	20	0.53	0.61	0.61	0.55	0.58
	30	0.66	0.82	0.71	0.76	0.74
	40	0.63	0.63	0.71	0.74	0.68
	50	0.68	0.61	0.68	0.76	0.68
Dynamic red.	5	0.32	0.32	0.63	0.50	0.44
	10	0.47	0.29	0.61	0.45	0.46
	20	0.53	0.66	0.53	0.47	0.55
	30	0.61	0.76	0.68	0.74	0.70
	40	0.61	0.61	0.61	0.76	0.65
	50	0.66	0.55	0.71	0.76	0.67
		1	2	3	4	
		RNG seed				

Table 7.1: Results from using objects from the whole range of speeds both in the training set and in the test set.

7.3 Using Data from the Region Around the Optimal Operating Point

Experiments using data from the region around the optimal operating point both in the training set and in the test set were carried out in almost the same manner as the experiments using objects from the whole range of speeds. However, step 1 is slightly different:

1. A Perl script, which can be seen in appendix A.3, was used to extract the objects situated in the region around the optimal operating point. Thus a subtable of the original decision table was created. This table was imported into ROSETTA and divided into a training set and a test set (10% - 90%).

2 - 8. These steps were carried out in the same manner as for the experiments using objects from the whole range of speeds.

The results from these experiments can be viewed in Table 7.2.

7.4 Combining the two Methods

We also used a classifier based on objects from the region around the optimal operating point to classify objects from the whole range of speeds. In this case a similar experiment as the one explained in section 7.3 was carried out. However, now the extraction of the objects situated in the region around the optimal operating point was done after the original decision table was divided into a training set and a test set. Thus the test set consisted of objects from the whole range of speeds while the training set only contained objects from the region around the optimal operating point. Only classifiers based on 30 objects from each decision class was considered. The results can be seen in Table 7.3.

	Number of objects from each decision class	Measure of accuracy				
		Split 1	Split 2	Split 3	Split 4	Mean value
Johnson alg.	5	0.55	0.50	0.41	0.50	0.49
	10	0.59	0.59	0.50	0.64	0.58
	20	0.59	0.82	0.68	0.86	0.74
	30	0.77	0.91	0.86	0.77	0.83
	40	0.73	0.91	0.82	0.77	0.81
	50	0.64	0.82	0.86	0.73	0.76
Genetic alg.	5	0.64	0.59	0.59	0.59	0.60
	10	0.73	0.68	0.73	0.73	0.72
	20	0.59	0.77	0.73	0.86	0.74
	30	0.73	0.86	0.95	0.77	0.83
	40	0.68	0.82	0.82	0.77	0.77
	50	0.68	0.82	1.00	0.73	0.81
Dynamic red.	5	0.68	0.59	0.59	0.64	0.63
	10	0.73	0.82	0.59	0.77	0.73
	20	0.68	0.91	0.73	0.86	0.80
	30	0.68	0.91	0.86	0.86	0.83
	40	0.64	0.91	0.91	0.82	0.82
	50	0.68	0.91	0.86	0.77	0.81
		1	2	3	4	
		RNG seed				

Table 7.2: Results from using objects from the region around the optimal operating point both in the training set and in the test set.

7.5 Using Mean Value Objects

As an alternative to copying randomly selected objects when too few objects were available we also tried to use a mean value object and copy this object when too few objects were available. The mean value object was constructed as explained in section 6.3.

The experiment was carried out almost in the same manner as the one explained in section 7.3. That is, objects from the region around the optimal operating point were used both in the training and in the test set. However, this time the mean value object was used when too few objects were available in step 2. The modified Perl script written for this purpose can be seen in

	Number of objects from each decision class	Measure of accuracy				
		Split 1	Split 2	Split 3	Split 4	Mean value
Johnson alg.	30	0.58	0.61	0.63	0.66	0.62
Genetic alg.	30	0.66	0.76	0.74	0.87	0.76
Dynamic red.	30	0.55	0.58	0.61	0.74	0.62
		1	2	3	4	
		RNG seed				

Table 7.3: Results from using objects from the region around the optimal operating point in the training set and objects from the whole range of speeds in the test set

appendix A5. Again only classifiers based on 30 objects from each decision class was tested. The result can be viewed in Table 7.4.

	Number of objects from each decision class	Measure of accuracy				
		Split 1	Split 2	Split 3	Split 4	Mean value
Johnson alg.	30	0.64	0.68	0.82	0.82	0.74
Genetic alg.	30	0.64	0.82	0.86	0.77	0.77
Dynamic red.	30	0.77	0.73	0.91	0.77	0.80
		1	2	3	4	
		RNG seed				

Table 7.4: Results from using objects from the region around the optimal operating point both in the training set and in the test set. When too few objects were available, a mean value object was copied until enough objects were obtained.

Chapter 8

Discussion/Conclusion

In this chapter we will try to draw some conclusions based on the results outlined in chapter 7. Then we will compare the results with results obtained using other methods explained in chapter 3. Last, we will look into the future and try to point out what should be done next.

8.1 Evaluation of the Results

In Figure 8.1 a graphical representation of the data from Table 7.1 and 7.2 can be viewed. One should remember that these cases are based on objects from the same region both in the training set and in the test set. Accuracy is shown as a function of the amount of objects used from each decision class. Now, conclusions can be drawn along three different dimensions. First, classifiers based on objects from the region around the optimal operating point and classifiers based on objects from the whole range of speeds should be compared. Second, classifiers based on different amount of objects should be compared. Third, different algorithms for reduct computation should be compared.

As one could expect, classifiers based on objects from the region around the optimal operating point did considerably better than classifiers based on objects from the whole range of speeds. This clearly shows that domain knowledge can be incorporated as a part of ROSETTA. However, this knowl-

edge is not explicitly used as in other systems, for example systems based on fault matrixes. Rather, the knowledge is used implicitly through how the data is prepared and how ROSETTA is used. In this case we used knowledge about the special properties of the region around the optimal operating point to select objects with the best classificatory qualities.

Intuitively, the performance of a given classifier should increase when more knowledge, in this case more objects, are available. However, the curves in Figure 8.1 show that as the number of objects from each decision class exceed the 30 object mark, performance doesn't get better. Rather, it gets slightly worse. This could indicate that too much information confuses the classifier, and that 30 objects from each decision class is the optimal number of objects to use. However, we should remember that a limited number of objects were available in the first place. Thus data sets containing more than 30 objects from each decision class includes a considerable number of identical objects. It is therefore more reasonable to believe that this phenomenon is caused by the fact that there is no new information in copied objects.

It is a clear tendency that the Johnson algorithm obtain poorer accuracy for 5 and 10 objects from each decision class, while it competes with the other algorithms when more objects are used. As we remember from chapter 5, the Johnson algorithm computes only one reduct from each discernibility function while the Genetic algorithm computes several. Thus it is reasonable to believe that the great difference in accuracy when a small number of objects is used is due to the fact that the Genetic algorithm in some sense extract more knowledge from each object, and that this property is more visible when few objects are available. For a larger number of objects the Johnson algorithm is more attractive than the Genetic algorithm, since the former computes a considerable smaller rule set. Thus it is easier for a domain expert to examine the rule set induced by the Johnson algorithm. It is more difficult to draw any definite conclusions with respect to dynamic reducts. Compared to the other algorithms they do better for objects collected from the region around the optimal operating point than they do for objects collected from the whole range of speeds. However, this difference is hardly significant.

In Figure 8.1 a vertical line is drawn where two thirds of the objects in the test set are correctly classified. This is due to the fact that domain experts on the field of rotating machinery consider this to be a very good result. Classifiers based on objects from the region around the optimal operating point classify

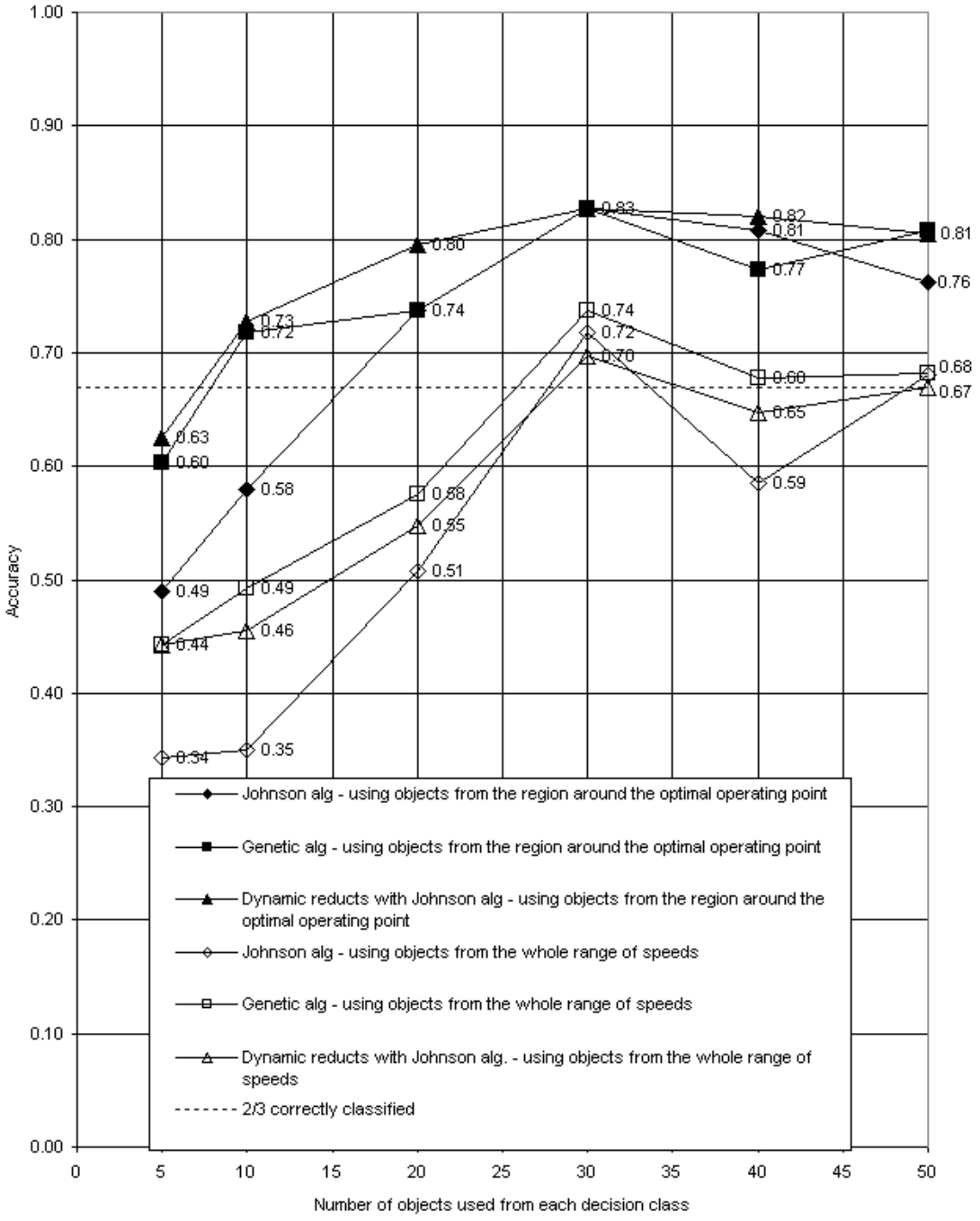

Figure 8.1: A graphical representation of the data from Table 7.1 and 7.2

with an accuracy considerably better than 0.67 when more than 5 objects from each decision class are used. Accuracy better than 0.67 is also obtained using objects from the whole range of speeds, but only when 30 or more objects are used from each decision class. Based on this experiment and these results one can thus conclude that ROSETTA is a strong tool for fault diagnosis in rotating machinery.

As stated in chapter 2, the main hypothesis behind machine learning methods is that the data in the training set is representative for the given domain. Thus one would expect a rather poor result when objects from the region around the optimal operating point are used to induce rules which in turn are used to classify objects from the whole range of speeds. However, Table 7.3 shows that this method actually did rather well. In particular, the Genetic algorithm obtained an accuracy of 0.76, which is slightly better than what was obtained using objects from the whole range of speeds both in the training and in the test set (see Table 7.1 or Figure 8.1). This could indicate that the rules induced from the region around the optimal operating point include general knowledge which is independent of speed and load.

The last experiment were done using a mean value object instead of coping randomly selected objects when too few objects were available. Table 7.4 shows that this approach didn't performed quite as good as copying random objects (See Table 7.2 or Figure 8.1). However, the experiment indicates that this approach is a rather good alternative.

We can now summarise the above discussion by turning our attention back to the questions asked in the beginning of chapter 6. It is shown that a training set consisting of 30 objects from each decision class is enough to classify data from the given diesel engine with good results. It is also shown that using data from the region around the optimal operating point not only reduces the amount of data one have to consider, but also gives considerably better results. This is also favourable with respect to the fact that the engine normally works in this region. Thus one can constantly measure the state of the engine without interfering with its operation.

8.2 Comparison to other Methods

It is difficult to compare results obtained in this projects with results obtained using other methods. In particular, a direct comparison is impossible unless all methods are used to diagnose exactly the same faults on the same machine. Since the machines diagnosed with VIBEX, inverse modelling with neural networks and ROSETTA are completely different, we have to settle with some general comments.

Domain experts on fault diagnosis in rotating machinery is often more concerned with the likelihood of a fault rather than the absolute accuracy. Thus in many systems the accuracy is not very good (around 0.50), but the results give an indication of which fault may be present. How ROSETTA can be used in the same manner is discussed later.

In [3] there is a discussion of quality of results using inverse modelling and neural networks. In this particular example, the confidence interval for the output from the neural network could be set as an error of ± 0.2. Thus for "large faults" with an expected output of 0.85 one should expect an accuracy better than 0.76 ($\frac{0.85-0.2}{0.85}$), for "small faults" with an expected output of 0.30 an accuracy better than 0.33 ($\frac{0.30-0.2}{0.30}$) and for "very small faults" with an expected output smaller than 0.2 no classification is made.

One advantage ROSETTA has over methods like neural networks is its capability of being more than just a classification tool. In other words, ROSETTA is considered a white box rather than, as is the case for neural networks, a black box. This gives the analyst the opportunity both to learn more about the data and to validate the extracted knowledge.

8.3 Further Work

During the project work, new problems have occurred and new insight has been gained. We will therefore describe some of the work we think needs to be done in the future:

- **Testing:** A new commando-line version of ROSETTA as been released. This will make it easier to automate experiments in the future. What

this project concerns, more splits should have been used in order to draw a more definite conclusion.

- **Reducing the decision table:** It is reasonable to believe that several of the existing attributes in the used decision table are more or less redundant. This hypothesis is not thoroughly tested and can thus not be establish with certainty. However, preliminary results shows that the decision table can be reduced to six or seven attribute without much loss of accuracy.

- **Pruning the rule sets:** Using 30 objects from each decision class, the number of rules induced for the different algorithms have approximately the following distribution:

Algorithm	Number of rules
Johnson alg.	50
Dynamic reducts with Johnson alg.	350
Genetic alg.	2000

  These numbers could be further reduced by using some algorithm for rule pruning. A rule set containing below 20 rules could in turn be validated by a domain expert in order to learn more both about the given data set and about how ROSETTA behave given this data set.

- **Calibration:** As mentioned in section 8.2, domain experts are often more concerned with the likelihood of a fault rather than the absolute classification. Thus experiments should be done with respect to calibration rather than discrimination. In chapter 5 it is explained that calibration is a measure of how close the output of the voting process, $\mathit{certainty}(x, \beta)$, is to the probability that object x belongs to decision class β. In ROSETTA this information is available in a log file, thus a simple script scanning this file could present this information to the user.

- **AUC:** As a measure of discrimination, AUC have many advantages over accuracy. One of them is its independency concerning the distribution of the amount of data from each decision class. Unfortunately, AUC can only be used in relation with a binary classifier. Thus in our case a pipeline of binary classifiers as shown in Figure 8.2 must be

created. Each classifier indicates whether an object belongs to a given decision class or not. The test set is classified six times, and possible inconsistency is cleared up using a voting algorithm. In this way an individual threshold can be used for each decision class and one could therefore expect a better classification altogether.

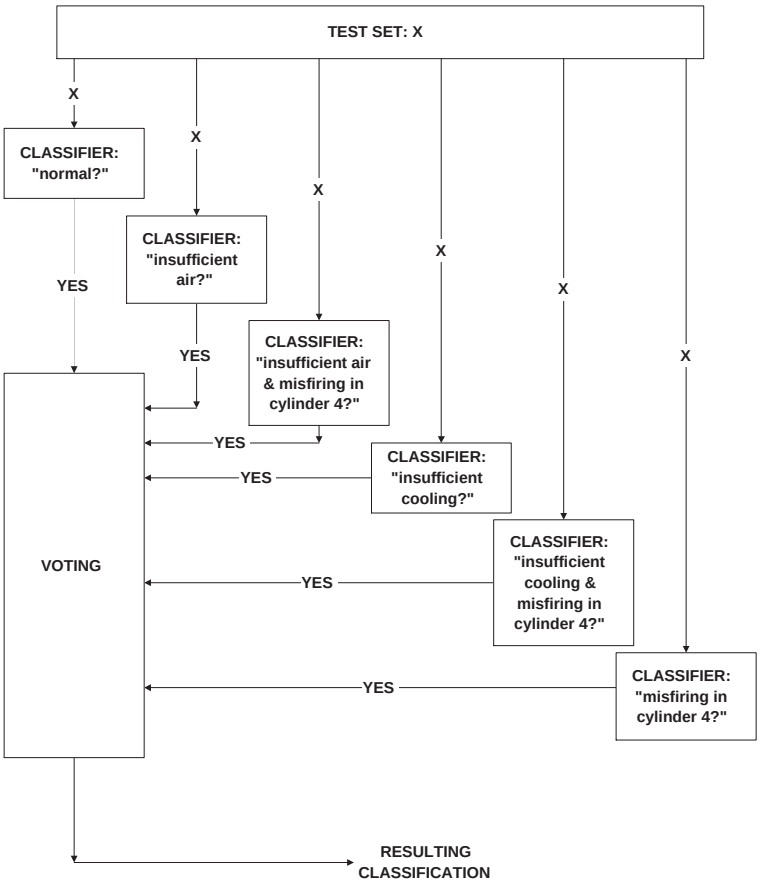

Figure 8.2: Using binary classifiers

Bibliography

[1] Torulf Mollestad. *A rough set approach to data mining: Extracting a logic of default rules from data*. PhD thesis, University of Trondheim, The Norwegian Institute of Technology, 1997.

[2] Maurice F. White. Expert systems for fault diagnosis of machinery. *Measurement, Journal of the International Measurement Confederation*, Vol9, No4, 1991.

[3] E. Cholewa and M. F. White. Inverse modelling in rotordynamics for identification of unbalance distribution. *Machine Vibration, Springer Verlag*, Vol. 2, No. 3, 1993.

[4] S.Russel and P. Norvig. *Artificial intelligence - a modern approach*. Prentice-Hall International, Inc., 1995.

[5] T.M.Mitchell. *Machine Learning*. McGraw-Hill Companies, Inc., 1997.

[6] E. Lihovd and M. Rasmussen. Neural networks in conditioning monitoring and diagnosis on rotating machinery. *10th Ship Control Systems Symposium in Ottawa, Canada*, 1993.

[7] C. Steinebach. *Knowledge based systems for diagnosis of rotating machinery*. PhD thesis, University of Trondheim, The Norwegian Institute of Technology, 1993.

[8] C. Steinebach and M. F. White. Vibex (part1) - an intelligent knowledge based system for fault diagnosis in turbomachinery. *NATO ASI Series, Vibration and Wear in High Speed Rotating Machinery, Ed. J.M. Montalvao e Silva & F.A. Pina da Silva, Kluwer Academic Publishers*, Vol. 174:759 – 772, 1990.

[9] Jan Komorowski, Zdzislaw Pawlak, Lech Polkowski, and Andrzej Skowron. A rough set perspective on data and knowledge.

[10] Alexander Øhrn. *ROSETTA, Technical Reference Manual*. http://www.idi.ntnu.no/∼aleks/rosetta, 1999.

Appendix A

Preprocessing scripts

A.1 konvert.prl

Data from the computer based measurement system connected to the diesel engine comes in binary files. A program exists, *konvert.exe*, which converts one binary file at a time into a plain text file. The following Perl script creates a program, *KONV.BAT*, which in turn can be run in Windows NT and which automatically converts every binary file in the directory.

```
#!/store/bin/perl

# Reads current directory and creates an IFOF.0XX for each BGL0216.0XX-file.
# Adds "cat IFOF.0XX | KONVERT.EXE" in KONV.BA.
# Run KONV.BAT under NT to convert the files (double click the bat-file icon).

opendir(DIRH, ".") || die "opendir failed";

# Change this line if the files don't start with "BGL"
@bglfiles = grep(/^BGL/, readdir(DIRH));

open(BAT, ">KONV.BAT") || die "open failed";

foreach (@bglfiles) {
    tr/A-Z/a-z/;
```

```
    ($firstname, $lastname) = split(/\./);

#   $lastname = $lastname +50;

    open(IFOF, ">IFOF.$lastname");
    print IFOF "$_\n$lastname.txt";
    close(IFOF);

    print BAT "type IFOF.$lastname | KONVERT.EXE
\n";
}
print BAT "del IFOF*";
close(BAT);
```

A.2 make_table.prl

Each object is situated in a separate plain text file. This Perl script builds a decision table from these text files.

```
#!/local/bin/perl

# Converts data from files delivered by Division of Marine
# Engineering into a format which Rosetta can read.

@directory = (normal,in_air,in_air_misfiring,in_cooling,
in_cool_misfiring,misfiring);

@dir0 = readpipe "ls $directory[0]/*TXT";
@dir1 = readpipe "ls $directory[1]/*TXT";
@dir2 = readpipe "ls $directory[2]/*TXT";
@dir3 = readpipe "ls $directory[3]/*TXT";
@dir4 = readpipe "ls $directory[4]/*TXT";
@dir5 = readpipe "ls $directory[5]/*TXT";

@dir = ([@dir0],[@dir1],[@dir2],[@dir3],[@dir4],[@dir5]);
```

```
@diagnosis = ("normal","\"insufficient air\"",
"\"insufficient air & misfiring in cylinder 4\"",
"\"insufficient cooling\"",
"\"insufficient cooling & misfiring in cylinder 4\"",
"\"misfiring in cylinder 4\"");

@diagnosis = (1,2,3,4,5,6);

open (DATA, $dir0[0]);
open (NEWDATA, ">Test/data.txt");

# Read attributes from file
# -------------------------

for ($i = 1; $i < 13; $i++) {
   readline *DATA;
}

for ($i = 1; $i <15; $i++) {
   read (DATA, $dummy, 2);
   read (DATA, $attr[$i], 25);
   readline *DATA;
}

readline *DATA;
read (DATA, $dummy, 2);
read (DATA, $attr[15], 25);
readline *DATA;

# Write attributes and formats to file
# ------------------------------------

for ($i = 1; $i <16; $i++) {
   print NEWDATA "\"$attr[$i]\"";
}
print NEWDATA "Diagnosis";
print NEWDATA "\n";
```

```
for ($i = 1; $i <16; $i++) {
   print NEWDATA "float(4)";
   print NEWDATA "    ";
}
print NEWDATA "string";

close (DATA);

# Write objects to file
# ---------------------

for ($k = 0; $k < ($#diagnosis+1); $k++) {        # For each diagnosis

   for ($j = 0; $j < ($#{$dir[$k]}+1); $j++) {    # For each object

      open(DATA, @{$dir[$k]}[$j]);

      for ($i = 1; $i < 13; $i++) {
         readline *DATA;
      }

      for ($i = 1; $i < 15; $i++) {
         read (DATA, $dummy, 37);
         read (DATA, $value[$i], 10);
         readline *DATA;
      }

      readline *DATA;
      read (DATA, $dummy, 37);
      read (DATA, $value[15], 10);
      readline *DATA;

      # Copying objects with rare diagnosis
      $copy = 1;
      if ($k == 0) {$copy = 1;}          # diagnosis = normal
      if ($k == 1) {$copy = 1;}          # diagnosis = in_air
      if ($k == 2) {$copy = 1;}          # diagnosis = in_air_misfiring
      if ($k == 3) {$copy = 1;}          # diagnosis = in_cooling
```

```
      if ($k == 4) {$copy = 1;}          # diagnosis = in_cool_misfiring
      if ($k == 5) {$copy = 1;}          # diagnosis = misfiring

      for ($c = 1; $c < ($copy+1); $c++) {

          print NEWDATA "\n";
          for ($i = 1; $i <16; $i++) {
              print NEWDATA $value[$i];
              print NEWDATA "    ";
          }

          # Diagnosis
          print NEWDATA $diagnosis[$k];
      }
      close (DATA);
   }
}
close (NEWDATA);
```

A.3 operate30.prl

This Perl script creates a subtable of the original decision table containing only objects from the region around the optimal operating point.

```
#!/local/bin/perl

# Reads all objects for one fault class and finds
# those objects which is within 30% of maximum operating point.

@tables = ("Test/table1.txt","Test/table2.txt","Test/table3.txt",
           "Test/table4.txt","Test/table5.txt","Test/table6.txt");

open (OPERATE30, ">Test/operate30.txt") || die "open failed";

for ($f = 0; $f < ($#tables + 1); $f++) {
```

```
open (DATA, $tables[$f]) || die "open failed";

@line = <DATA>;

if ($f == 0) {
    print OPERATE30 $line[0];
    print OPERATE30 $line[1];
}

$maxload = 0;
$minspeed = 999999999999;
for ($i=2; $i < ($#line+1); $i++) {
    ($attrib1[$i-2],$attrib2[$i-2],$attrib3[$i-2],$attrib4[$i-2],
     $attrib5[$i-2],$attrib6[$i-2],$attrib7[$i-2],$attrib8[$i-2],
     $attrib9[$i-2],$attrib10[$i-2],$attrib11[$i-2],$attrib12[$i-2],
     $attrib13[$i-2],$attrib14[$i-2],$attrib15[$i-2],$attrib16[$i-2])
        = split /\s+/,$line[$i],16;

    if ($maxload < $attrib2[$i-2]) {
        $maxload = $attrib2[$i-2];
        $max_opr_speed = $attrib1[$i-2];
    }
    if ($minspeed > $attrib1[$i-2]) {$minspeed = $attrib1[$i-2];}
}

@data = ([@attrib1],[@attrib2],[@attrib3],[@attrib4],
         [@attrib5],[@attrib6],[@attrib7],[@attrib8],
         [@attrib9],[@attrib10],[@attrib11],[@attrib12],
         [@attrib13],[@attrib14],[@attrib15],[@attrib16]);

$decision = $f+1;
print "$decision: Max_opr_speed: $max_opr_speed\n";

$limit = $max_opr_speed-0.3 * ($max_opr_speed - $minspeed);

print "$decision: Limit: $limit\n";

for ($i=0; $i < ($#line-1); $i++) {
```

```
        if (@{$data[0]}[$i] > $limit) {
            for ($j=0; $j < (15); $j++) {
                print OPERATE30 @{$data[$j]}[$i];
                print OPERATE30 "    ";
            }
                print OPERATE30 @{$data[15]}[$i];
        }
    }
    print "$tables[$f]\n";
    print OPERATE30 "\n";
}
```

A.4 duplicate.prl

This Perl script converts a decision table into a new decision table with a given number of objects from each decision class.

```
#!/local/bin/perl

# Reads training data and duplicate selected objects.

#$NUMBER = $ARGV[0];
@NUMBER = ($dummy,20,20,20,20,20,20);

open (TRAIN, "Test/train.txt") || die "open failed";
open (DUPTRAIN, ">Test/duptrain.txt") || die "open failed";

@line = <TRAIN>;

print DUPTRAIN $line[0];
print DUPTRAIN $line[1];

@faultstart = ($dummy,1,101,201,301,401,501);
@fault      = ($dummy,0,0,0,0,0,0);

@data = ($dummy,[@A1],[@A2],[@A3],[@A4],[@A5],[@A6],[@A7],[@A8],[@A9],
```

```
      [@A10],[@A11],[@A12],[@A13],[@A14],[@A15],[@D]);

@attrib;

# Read data from file

for ($i = 2;$i < ($#line+1);$i++) {
    ($attrib[1],$attrib[2],$attrib[3],$attrib[4],$attrib[5],$attrib[6],
     $attrib[7],$attrib[8],$attrib[9],$attrib[10],$attrib[11],$attrib[12],
     $attrib[13],$attrib[14],$attrib[15],$attrib[16])
    = split /\s+/,$line[$i],16;

    if ($attrib[16] == 1) {$index = $fault[1]+$faultstart[1]; $fault[1]++;}
    if ($attrib[16] == 2) {$index = $fault[2]+$faultstart[2]; $fault[2]++;}
    if ($attrib[16] == 3) {$index = $fault[3]+$faultstart[3]; $fault[3]++;}
    if ($attrib[16] == 4) {$index = $fault[4]+$faultstart[4]; $fault[4]++;}
    if ($attrib[16] == 5) {$index = $fault[5]+$faultstart[5]; $fault[5]++;}
    if ($attrib[16] == 6) {$index = $fault[6]+$faultstart[6]; $fault[6]++;}

    for ($j=1; $j <17; $j++) {
        @{$data[$j]}[$index] = $attrib[$j];
        }
}

# Duplicate objects in decision classes with
# less than $NUMBER objects

for ($i = 1; $i < 7; $i++) {
    if ($fault[$i] < $NUMBER[$i]){
        for ($j = 1; $j < ($NUMBER[$i] + 1 - $fault[$i]); $j++) {
            for ($k = 1; $k < 17; $k++) {
                @{$data[$k]}[$faultstart[$i]+$fault[$i]+$j-1] =
                    @{$data[$k]}[$faultstart[$i]+$j-1];
            }
        }
    }
}
```

```
# Write data to file

for ($i = 1; $i < 7; $i++) {
    for ($j = 1; $j < ($NUMBER[$i]+1); $j++) {
        for ($k = 1; $k < 17; $k++) {
            print DUPTRAIN @{$data[$k]}[$faultstart[$i]+$j-1];
            if ($k != 16){print DUPTRAIN "    ";}
        }
    }
}
```

A.5 mean_duplicate.prl

This Perl script works in the same way as *duplicate.prl*, except that when too few objects from a given decision class exist, a mean value object is copied instead of a randomly selected object.

```
#!/local/bin/perl

# Reads training data and duplicate selected objects.

#$NUMBER = $ARGV[0];
@NUMBER = ($dummy,30,30,30,30,30,30);

open (TRAIN, "Test/train.txt") || die "open failed";
open (DUPTRAIN, ">Test/duptrain.txt") || die "open failed";

@line = <TRAIN>;

print DUPTRAIN $line[0];
print DUPTRAIN $line[1];

@faultstart = ($dummy,1,101,201,301,401,501);
@fault      = ($dummy,0,0,0,0,0,0);

@data = ($dummy,[@A1],[@A2],[@A3],[@A4],[@A5],[@A6],[@A7],[@A8],[@A9],
```

```
      [@A10],[@A11],[@A12],[@A13],[@A14],[@A15],[@D]);

@mean = ($dummy,[@A1],[@A2],[@A3],[@A4],[@A5],[@A6],[@A7],[@A8],[@A9],
      [@A10],[@A11],[@A12],[@A13],[@A14],[@A15],[@D]);
for ($i = 1;$i < 17; $i++) {
    for ($j = 1;$j < 7; $j++) {
        @{$mean[$i]}[$j] = 0;
    }
}

@attrib;

# Read data from file

for ($i = 2;$i < ($#line+1);$i++) {
    ($attrib[1],$attrib[2],$attrib[3],$attrib[4],$attrib[5],$attrib[6],
     $attrib[7],$attrib[8],$attrib[9],$attrib[10],$attrib[11],$attrib[12],
     $attrib[13],$attrib[14],$attrib[15],$attrib[16])
    = split /\s+/,$line[$i],16;

    if ($attrib[16] == 1) {$index = $fault[1]+$faultstart[1]; $fault[1]++;
                           $f = 1}
    if ($attrib[16] == 2) {$index = $fault[2]+$faultstart[2]; $fault[2]++;
                           $f = 2}
    if ($attrib[16] == 3) {$index = $fault[3]+$faultstart[3]; $fault[3]++;
                           $f = 3}
    if ($attrib[16] == 4) {$index = $fault[4]+$faultstart[4]; $fault[4]++;
                           $f = 4}
    if ($attrib[16] == 5) {$index = $fault[5]+$faultstart[5]; $fault[5]++;
                           $f = 5}
    if ($attrib[16] == 6) {$index = $fault[6]+$faultstart[6]; $fault[6]++;
                           $f = 6}

    for ($j=1; $j <17; $j++) {
        @{$data[$j]}[$index] = $attrib[$j];
        @{$mean[$j]}[$f] = @{$mean[$j]}[$f] + $attrib[$j];
        }
}
```

```
#Compute mean value objects

for ($i = 1;$i < 17; $i++) {
    for ($j = 1;$j < 7; $j++) {
        @{$mean[$i]}[$j] = @{$mean[$i]}[$j]/$fault[$j];
    }
}

for ($i = 1;$i < 7; $i++) {
@{$mean[16]}[$i] = "$i\n";
}

# Duplicate objects in decision classes with
# less than $NUMBER objects

for ($i = 1; $i < 7; $i++) {
    if ($fault[$i] < $NUMBER[$i]){
        for ($j = 1; $j < ($NUMBER[$i] + 1 - $fault[$i]); $j++) {
            for ($k = 1; $k < 17; $k++) {
                @{$data[$k]}[$faultstart[$i]+$fault[$i]+$j-1] =
                    @{$mean[$k]}[$i];
            }
        }
    }
}

# Write data to file

for ($i = 1; $i < 7; $i++) {
    for ($j = 1; $j < ($NUMBER[$i]+1); $j++) {
        for ($k = 1; $k < 17; $k++) {
            print DUPTRAIN @{$data[$k]}[$faultstart[$i]+$j-1];
            if ($k != 16){print DUPTRAIN "    ";}
        }
    }
}
```