Application of Rough Sets Algorithms to Prediction of Aircraft Component Failure

José M. Peña[2], Sylvain Létourneau[1], and Fazel Famili[1]

[1] Institute for Information Technology, National Research Council, Ottawa, Canada
{sylvain.letournau, fazel.famili}@iit.nrc.ca
[2] Department of Computer Science, Universidad Politécnica de Madrid, Spain

Abstract. This paper presents application of *Rough Sets* algorithms to prediction of component failures in aerospace domain. To achieve this we first introduce a data preprocessing approach that consists of case selection, data labeling and attribute reduction. We also introduce a weight function to represent the importance of predictions as a function of time before the actual failure. We then build several models using rough set algorithms and reduce these models through a postprocessing phase. End results for failure prediction of a specific aircraft component are presented.

1 Introduction

Rough Sets theory was first defined by Pawlak [10,11]. During the last few years it has been applied in Data Mining and Machine Learning environments to different application areas [9,7]. As demonstrated by these previous applications and its formalized mathematical support, *Rough Sets* are efficient and useful tools in the field of knowledge discovery to generate discriminant and characteristic rules. However, in some cases the use of this technique and its algorithms requires some preprocessing of the data. In this paper, we explain the application of the *Rough Sets* algorithms and the preprocessing involved in order to use these techniques for prediction of component failures in the aerospace domain.

In today's aerospace industry the operation and maintenance of complex systems, such as commercial aircraft is a major challenge. There is a strong desire to monitor the entire system of the aircraft and predict when there is a potential for certain components to fail. This is specially true when in modern aircraft there is access to complex sensors and on-board computers that collect huge amounts of data at different stages of operation of the aircraft and transmit this data to ground control center where it is available in real-time. This information usually consists of both text and parametric (numeric/symbolic) data and it exceeds 2-3 megabytes of data per month for each modern aircraft. In most cases this data may not be used or even properly warehoused for future access. Several reasons exist: (i) engineers and operators do not have sufficient time to analyze huge amounts of data, unless there is an urgent requirement, (ii) complexity of the data analysis process is in most cases beyond the ordinary

D.J. Hand, J.N. Kok, M.R. Berthold (Eds.): IDA'99, LNCS 1642, pp. 473–484, 1999.

tools that they have, and (iii) there is no well defined automated mechanism to extract, preprocess and analyze the data and summarize the results so that the engineers and technicians can use it.

Several benefits could be obtained from proper prediction of component failures. These are: (i) reducing the number of delays, (ii) reducing the overall maintenance costs, (iii) potential increase in safety, and (iv) preventing additional damage to other components.

The data used in this research comes from automatically acquired sensor measurements of the auxiliary power units (APU) of 34 Airbus A320 aircraft. This data has been acquired between 1994-97 and it consists of two major parts: (i) all repair actions taken on these aircraft, and (ii) all parametric data acquired during the operation of these power units. Examples of problems with this data were: missing attributes, out-of-range attributes and improper data types. After cleaning the original data, a data set consisting of about 42000 cases was prepared.

Our goal was to use this data to generate models (in the form of rules) that explain failure of certain components. These rules would then be used in a different system in order to monitor the data and generate alerts and inform the user when there is a potential for certain components to fail. This paper explains the process and the results of our research for the use of *Rough Sets* in prediction of component failures. In Section 2 we provide an overview of the approach. Section 3 includes the data preprocessing procedure and in Section 4 we explain the process of building a model. Section 5 contains the results and Section 6 is conclusion and future work.

2 Overview of the Approach

The aim of the rule extraction process described in this paper is to generate a valid set of prediction rules for aircraft component failures. These rules will have to accurately recognize particular patterns in the data that indicate an upcoming failure of a component.

The rule inference process starts by the selection of the data related to the component of interest. This is done in two steps. First, we retrieve, from the historical maintenance reports, the information about all occurrences of failure of the given component. The information retained is the failure dates along with the identifiers of the aircraft (or engine) on which the failures happened. Then we use this information to retrieve all the sensor measurements observed during the preceding days (or weeks) of each failure event. We also keep some data obtained during the days following the replacement of the component. Two new attributes are added to the initial raw measurements: the time between the observation is collected and the actual failure event, and a tag identifying each observation to a specific failure case. The data from all failures are finally combined to create the dataset used to build the predictive model.

In order to use a supervised learning approach such as *Rough Sets* algorithms as well as many others [13,12], we must add another attribute to the dataset

just created. That is the CLASS (or LABEL) attribute. The algorithm used to generate this new attribute is also called *labeling algorithm*.

In our case, the labeling algorithm creates a new attribute with two different values (0 and 1). This new attribute is set to 1 for all cases obtained between the time of the failure and the preceding n days (these n days define the window that we target for the failure predictions), and set to 0 for all other cases observed outside that period of time. Following the labeling of the data, some data preprocessing is performed which is explained in Section 3.

The next step is to build the models. This includes: selection of the relevant attributes, execution of *Rough Sets* algorithms, and post-processing of the results. Finally, the end results are evaluated. The overall process is summarized in Figure 1.

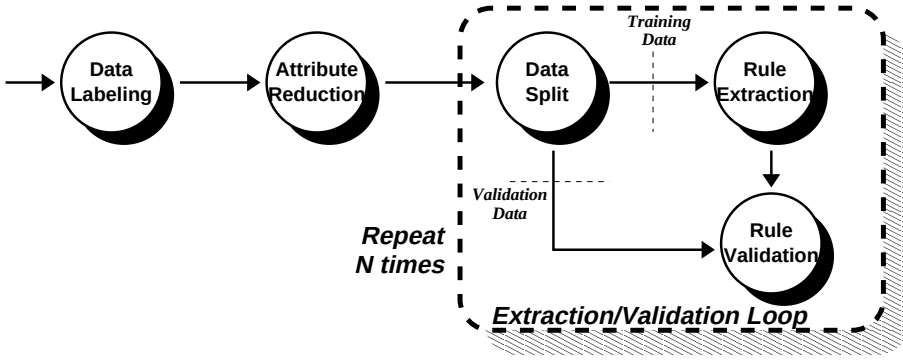

Fig. 1. General rule extraction procedure.

3 Data Preprocessing

This section explains preprocessing steps required before the application of the *Rough Sets* algorithms.

3.1 Discretization Algorithm

One of the requirements of all standard *Rough Sets* algorithms is that the attributes in the input data table need to be discrete (also known as nominal attributes). However, in the aerospace domain, the sensored data usually consists of continuous attributes and therefore a discretization process is required.

Discretization algorithms can be classified by two different criterion. The first division of these techniques is between *local* or *global* algorithms. *Local* algorithms are considered as some form of an induction algorithm (like C4.5 [13]). These algorithms perform partitions that are applied in some iterations of the

induction process such as in a number of nodes during tree construction. *Global* algorithms are used to transform continuous attributes into nominal attributes in a preliminary preparation task and with no direct interaction with the subsequent analysis processes. The second classification of discretization techniques defines *supervised* and *unsupervised* methods. *Supervised* algorithms use label (or class) information to guide discretization process and *unsupervised* methods apply different kinds of discretization criteria (such as equal interval width or equal frequency intervals).

In our experiments, we have discarded *local* methods because: (1) *global* algorithms are less prone to variance in estimation from small data size (some experiments [3] with C4.5 have been improved using preliminary global discretizations before C4.5 induction with no local discretization) and (2) our rule extraction process is performed by *Rough Sets* algorithms that require the previous discretization. We have chosen *supervised* techniques because using classification information we can reduce the probability of grouping different classes in the same interval [8]. Some typical *global supervised* algorithms are: *ChiMerge* [8], *StatDisc* [14] (both of them use statistical operators as part of the discretization function), *D-2* (entropy-based discretization [2]), and *MCC* (find partition boundaries using contrast functions [16]). But we have chosen *InfoMerge* [1], an information-theoretic algorithm, that substitutes *ChiMerge*/*StatDisc* statistical measures with an information loss function in a bottom-up iterative process. This approach is similar to C4.5 local discretization process but in order to apply it into a global algorithm a correction factor need to be used. This factor adjusts information function using interval weight (number of elements).

3.2 Weight Function

The second transformation operation is not so closely related to algorithm requirements and its application is motivated by a better rule quality at the end of the process. As described in Section 2, the labeling mechanism selects all the records in the last 30 days before the failure as positive data (the rules generated by the model will discriminate this time window from the data before and after this period). But the importance of the detection of this situation is not the same during all this period. For example, a component failure alert 20 days before the possible failure is less important than 5 days before and alerts too close to the failure do not allow any corrective actions. This domain characteristic can be described as a weight function as shown in the Figure 2. This weight function example defines three different values connected by a step function and it is an example of the distribution of the importance of alerts for this component. All algorithms of the procedure have been revised in order to use this weight function.

4 Building a Model

In this section, the three main steps of the model building phase are described in detail. These steps are: i) attribute reduction, ii) rules extraction, and iii)

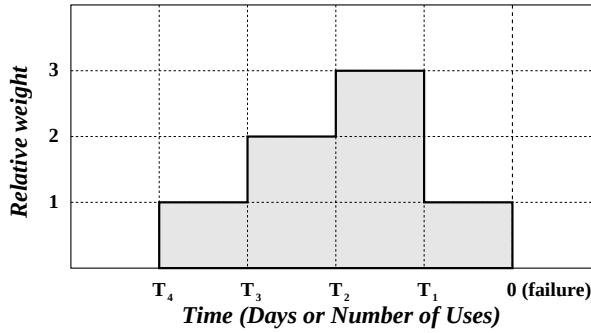

Fig. 2. Weight function example.

rules post-processing. In this research, *Rough Sets* algorithms have been used to implement each of these phases.

4.1 Attribute Reduction

In this phase of the process, we select from an original set of attributes, provided by the user, a subset of characteristics to use in the rest of the process. The selection criteria are based on the reduct concept description, as defined by [11]. The term REDUCT is defined as "*the essential part of knowledge, which suffices to define all basic concepts occurring in the considered knowledge*". In this problem's context we can define reduct as the reduced set of features that are able to predict the component failure.

Many different algorithms have been developed in order to obtain this reduced set of attributes [15,6]. Not all of them are suitable for our domain. For instance, the *Discernibility Matrix* algorithm [15] defines a triangular matrix with a size equal to the number of records in both dimensions. This algorithm would not be appropriate due to the size of the matrix it requires (e.g. for a problem of 20000 records it is necessary to handle a matrix of about 200 million cells). Another traditional method to calculate this set is to generate all combinations of attributes and then evaluate the classification power of each combination. The usual way to perform this evaluation is to calculate the *Lower* approximation [11]. *Lower* is a set of original records that belong to the concept and they are selected by an equivalence relation described by some attributes. These attributes are used to define this *Lower* region. If an element belongs to this approximation then it *surely* belongs to the class (the set of records we want to classify).

$$U:\ Universe\ (all\ the\ records). \tag{1}$$

$$X:\ Elements\ that\ belong\ to\ the\ CLASS\ (concept).\ X \subseteq U \tag{2}$$

$$R:\ Equivalence\ relation\ (defined\ by\ the\ attributes). \tag{3}$$

$$Lower = \{x \in U : [x]_R \subseteq X\} \tag{4}$$

In our experiments, we have used a simple reduct calculation algorithm. The main goal was not to obtain the minimal attribute reduct, but to provide a good result at a reasonable cost in terms of computation time and memory used. The algorithm implemented also uses the *Lower* approximation calculation [11] to evaluate the classification power of a set of attributes in each of the iterations. This approximation represents the set of data records successfully classified by a set of attributes. Therefore, the set of attributes is designed to preserve this original *Lower* region. The algorithm pseudo code is shown in Figure 3.

```
1:AttributeSet Calculate_Reduct(Data data,AttrSet attr)
2:{
3:  AttributeSet red={};
4:  Float acc,maxAcc=0.0,attrAcc[attr.size()];
5:  Attribute at1,at2,a,b;
6:
7:  while(maxAcc<REQUIRED_ACCURACY) {
8:    maxAcc=0.0;
9:    for(a in attr) {
10:     attrAcc[a]=Lower_Approximation(data,red+{a});
11:     for(b in attr) {
12:       acc=Lower_Approximation(data,red+{a,b});
13:       if(acc>=maxAcc) {
14:         maxAcc=acc;
15:         at1=a;
16:         at2=b;
17:       }
18:     }
19:     attr=attr-{a};
20:   }
21:   if(attrAcc[at1]>attrAcc[at2])
22:     red=red+{at1};
23:   else
24:     red=red+{at2};
25: }
26: return(red);
27:}
```

```
1:Float Lower_Approximation(Data data,AttributeSet attr)
2:Pre: "'data' must be sorted by 'attr'"
3:{
4:  Float pos=0.0,neg=0.0; cls=0.0; tot=0.0;
5:  Tuple reference,current;
6:
7:  reference=data.first();
8:  for(current in data) {
9:    if(IsEqual(current,reference,attr)) {
10:     if(IsPositive(current)
11:       pos+=current.weight;
12:     else
13:       neg+=current.weight;
14:   else {
15:     tot+=pos;
16:     if(pos/(pos+neg)>VPRSM_THRESHOLD) {
17:       cls+=pos;
18:       Write_Rule(reference,pos,pos+neg);
19:     }
20:     reference=current;
21:     if(IsPositive(current)
22:       pos=current.weight; neg=0.0;
23:     else
24:       neg=current.weight; pos=0.0;
25:   }
26: }
27: return(cls/tot);
28:}
```

Fig. 3. Non-optimal reduct calculation / Lower approximation calculation algorithms.

In each iteration, this algorithm first selects the best subset of two attributes based on the classification power (calculated with `Lower_Approximation`). It then selects the best attribute from these two. This algorithm is very efficient since it limits the search for the best subset of two attributes only. However, that limitation may also have an impact on the results obtained. It might be appropriate to run a modified version of this algorithm that can also search for the best subset of 3 attributes, or even more.

In Figure 4 there is a comparison between the combinatorial calculation of the reduct and the calculation using our approximative algorithm. The figure pictures the number of times Lower Approximation function has to be executed. For example, to calculate a 5-attribute reduct from 80 original attributes, with the combinational approach over 30 millions Lower regions must be calculated, but with the other algorithm there are only 13450 regions to calculate.

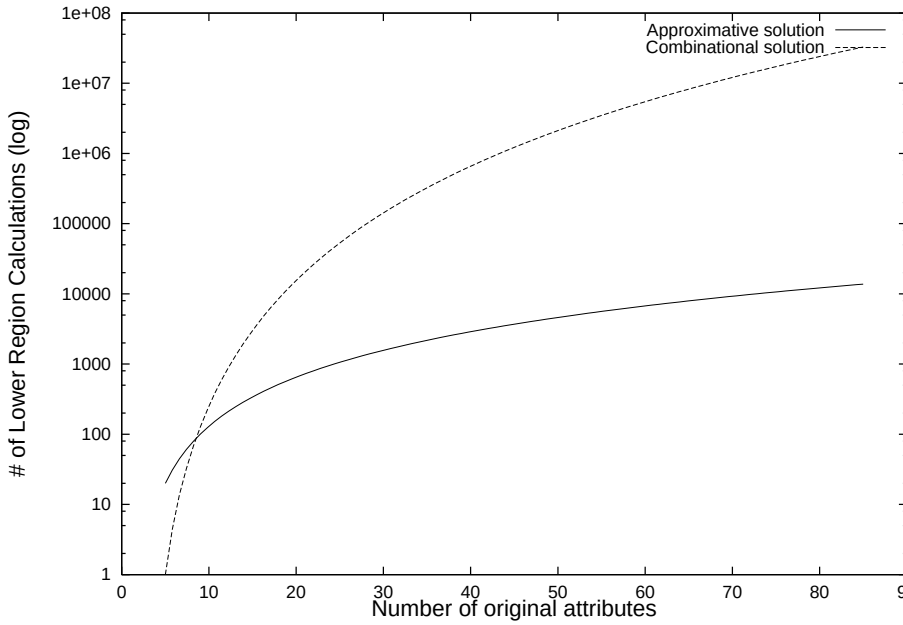

Fig. 4. Calculation of a 5-attribute reduct

4.2 Rule Extraction

At the core of the building model process we find the rule extraction step. The algorithm to perform that step scans the training data and extracts discriminant rules for the the target concept using the selected subset of attributes (obtained from the attribute reduction algorithm, see section 4.1). In our experiments, we have selected a fixed number of attributes for the reduct computation (the most discriminant ones, according to the reduct criteria). In other words, we forced the rule extraction algorithms to work with only a small subset of features. This constraint was necessary to limit the size of the rules generated and helped in keeping a good level of comprehensibility for domain experts that will have to review the results.

In our experiments, we also used *Lower* approximation calculation to generate the rules that describe the concept (i.e. the situations for which we should predict a specific component failure). Using this approach, each rule obtained consists of a conjunction of attribute value conditions (one condition per input attribute). As we will see in Section 4.3, this set of rules had to be processed before being used to predict component failure.

The implementation developed in our research supports Variable Precision Rough Set Model (VPRSM as defined by [17]) and the algorithm used is based on the design proposed by [5]. VPRSM extends traditional rough sets theory providing an inclusion threshold that allows more flexibility. With VPRSM an element x belongs to *Lower* region if more than $\alpha\%$ of elements in the same

equivalence class ($[x]_R$) belong to the concept. The only variation of this algorithm is related to the use of the weight function and its effect on threshold comparison process in VPRSM (see figure 3).

4.3 Rule Postprocessing

The number of rules obtained from the rule extraction process described above is typically very high. This section first explains why so many rules are generated and then, it explains an approach developed to transform the rule set obtained into a smaller one.

First, one of the characteristics of rules extracted by the *Lower* approximation calculation is that all the rules are expressed in terms of all the attributes provided to the algorithm. Each rule extracted using this technique is a conjunction of predicates. The format of these predicates is $attribute = value$, and all the attributes appear in all the rules. Clearly, with such a representation, the number of rules required to cover all possibilities is very large.

The quality of the discretization process may also have an impact on the total number of rules generated. Because the discretization process is independent of the rule extraction algorithm used, an attribute may be splitted into more intervals than required to generate the rules. In these cases, two or more rules are generated that only differ in the value of a discretized attribute and this two or more values represent consecutive intervals. Such a non optimal splitting of the attributes will contribute to enlarge the number of rules obtained.

In order to reduce the number of rules, a two-phase algorithm has been developed. In the first phase all the initial rules are combined to generate new rules, these new rules are more general (include all the elements described by both of the combined rules) than previous ones. This process is repeated until no new rule can be generated. In each of the iterations any initial or previously generated rules can be combined. In a second phase, all the rules that are described by a more general rule (all of the elements represented by the rule are also represented by another rule) are removed. The result of this second phase is a final set of rules equivalent to the original one but smaller (or in the worst case equal). This process cannot be achieved by a single combination/pruning phase since some rules may be used to generate more than one new rule. An example of execution of this algorithm is shown in Figure 5.

The final output of this algorithm is a smaller set of postprocessed more general rules. These rules are finally sorted by their support. The support being defined as the ratio between the number of cases in which this rule can be applied and the total number of cases.

5 Performance and Results

In this section, we report the results obtained by our approach to learn models to predict failure of the Auxiliary Power Unit (APU) starter motor. We also study the relationship between two important parameters of the approach. The process for our experiment is as follow:

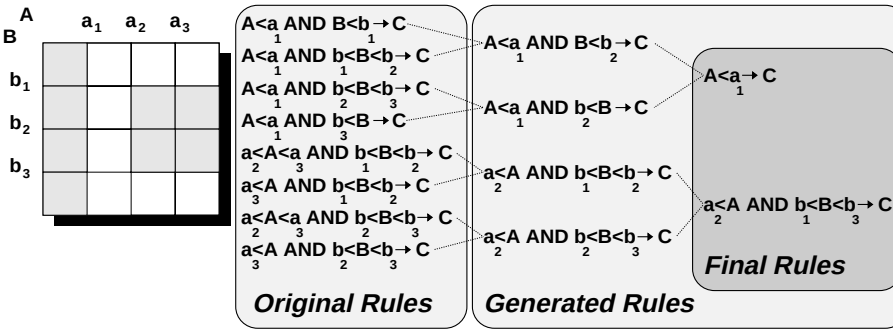

Fig. 5. Rule postprocessing example.

1. The data is splitted into batches. One batch being created for each failure case. For the APU starter problem, we had data from 30 failure cases (30 batches were then created).
2. We execute our approach to learn the rules using data form 29 cases and then use the data from the remaining case for validation. We repeat this step until data from each case has been used for validation (which means 30 iterations for the current component).
3. We use the validation results from the different runs to compute: (i)the number of cases for which we have at least one good alert generated during the prediction window(see Section 2), and (ii)the number of cases for which we have one ore more alerts generated outside the prediction window. In Table 1, these two numbers are referred to as Good Alert and False Alert, respectively.

We repeated the above process several times with different settings for two important parameters in our approach: the VPRSM threshold and the maximal number of intervals generated by the discretization algorithms. We experimented with VPRSM thresholds of .99, .97, .95, .90, and .80. Similarly, we experimented with values of $2, 3, 5, 7$, and 10 for the maximal number of discretization intervals. Table 1 presents the results from our experiments. The impact of these two parameters on the final results is very significant. In the top left side of the table, with high restrictive thresholds and a small number of intervals, the percentages of correct failure predictions and false alerts are both very low. On the other hand, low VPRSM thresholds and large number of intervals for discretization (bottom left corner of the table) lead to a high percentage of correct failures predictions along with an important ratio of false alerts. It is very interesting to note the impact of the maximal number of intervals for discretization. For instance, with a VPRSM threshold of .97, increasing the maximal number of intervals from 5 to 7 lead to an increase of 20% in the number of failures predicted and to a 26% decrease of the false alert ratio.

Finally, the most interesting result was obtained with a threshold of .97 and a maximal of 7 intervals. This result shows a good ability of the model in predicting

failures of the APU starter motor (70%) with a reasonable percentage of false alerts (6.7%).

Table 1. VPRSM threshold vs maximun number of intervals

Threshold	# Intervals	2	3	5	7	10
0.99	Good Alert:	3.3%	6.7%	20.0%	33.3%	26.7%
	False Alert:	10.0%	6.7%	10.0%	6.7%	10.0%
0.97	Good Alert:	3.3%	20.0%	50.0%	**70.0%**	23.3%
	False Alert:	10.0%	33.3%	33.3%	**6.7%**	10.0%
0.95	Good Alert:	6.7%	26.7%	40.0%	56.7%	40.0%
	False Alert:	16.7%	23.3%	10.0%	93.3%	33.3%
0.90	Good Alert:	10.0%	23.3%	63.3%	83.3%	86.7%
	False Alert:	16.7%	20.0%	43.3%	66.7%	96.7%
0.80	Good Alert:	10.0%	36.7%	70.0%	83.3%	93.3%
	False Alert:	16.7%	30.0%	66.7%	96.7%	96.7%

The rules extracted by our model never have more than five attributes (predicates). This rule size is close to the limit above which human comprehensibility becomes difficult. This characteristic is quite important because the predictive rules are processed by an automated monitoring tool that generates alerts with these rules and for each of the alerts the associated rule needs to be shown to an expert user who decides on corrective actions to be taken. An example of a rule obtained is:

```
IF 50.000<=SMIN15<52.000 AND 713.000<=EMIN20 AND 522.000<=EMAX
THEN "APU starter motor will fail within 15 days"
```

Similar rules can be generated by other algorithms. We are experimenting with other systems such as C4.5 and other algorithms accessible trough MLC++ [4]. Results obtained so far tend to show that the approach developed in this paper is competitive with well known decision tree systems in both the execution time and the accuracy of the results. For instance, the best model obtained so far with C4.5 has been able to correctly predict 77% of the failures with a false alert rate of about 9%. In terms of execution time, our *Rough Sets* implementation and C4.5 are also quite similar; each experiment for the selected component takes about 25 minutes with both systems.

6 Conclusions and Future Work

In this paper we present a new approach to the use of *Rough Sets* algorithm for prediction of component failures. Our data came from a real world aerospace application for which accurate predictions of component failures will be extremely useful. The approach consists of an extensive data reduction process, use of a

global supervised algorithm for discretization and a weight function to evaluate the performance of our experiments. The experiments carried out in our research revealed that the large number of rules generated by the algorithms had to be reduced to a smaller set for human comprehensibility. This was done using a novel approach that significantly reduces the number of rules without affecting the accuracy of the results.

An extensive experiment has been run to verify the impact of two parameters: the VPRSM threshold and the maximal number of intervals generated during discretization. The experiment has shown that the quality of the results is heavily affected by the maximal number of discretization intervals chosen. The experiment has also demonstrated that the overall approach is useful for obtaining rules that can predict up to 70% of the APU starter motor failures (prediction of the component targeted in this research) with a very reasonable rate of false alerts (less than 7%). This kind of models could lead to important savings for an airline.

The research framework described in this paper can be used as a basis for our future research in this area. Different discretization algorithms, weight functions and attribute reduction techniques along with other forms of rule postprocessing strategies can be experimented.

Acknowledgments

J.M. Peña thanks Dr. C. Fernández and Dr. E. Menasalvas for helpful comments and suggestions and also thanks all IIT group at NRC for support and help.

References

1. Alves Freitas, A. and Lavington, S. H.: Speeding up Knowledge Discovery in Large Relational Databases by Means of a New Discretization Algorithm. BNCOD 1996: 124–133.
2. Catlett, J.: On Changing Continuous Attributes into Ordered Discrete Attributes. EWSL 1991: 164–178.
3. Dougherty, J., Kohavi, R. and Sahami, M.: Supervised and Unsupervised Discretizations of Continuous Features. ML 1995: 194–202.
4. Kohavi, R., Sommerfield, D., and Dougherty, J.: Data Mining Using MLC++: A Machine Learning Library in C++. Tools with Artificial Intelligence. IEEE Computer Society Press 1996: 234–245.
5. Fernández-Baizán, C., Menasalvas, E. and Peña, J.M.: Rough Sets as a Foundation to Add Data Mining Capabilities to a RDBMS. CESA 1996: 764–769.
6. Fernández-Baizán, C., Menasalvas, E. and Peña, J.M.: A New Approach for the Calculation of Reducts in Large Databases. JICS 1997: 340–344.
7. Hadjimichael, M.: Discovering Fuzzy Relationships from Databases: CESA 1996: 830–835.
8. Kerber, R.: ChiMerge: Discretization of Numeric Attributes. AAAI 1992: 123–128.
9. Lin, T.Y.: Rough Set Theory in Very Large Databases. CESA 1996: 936–941.
10. Pawlak, Z.: Rough sets. IJICS 1982: 344–356.

11. Pawlak, Z.: Rough Sets - Theoretical Aspects of Reasoning about Data. Kluwer Ed., 1991.
12. Quinlan, J.R.: Induction of Decision Trees. ML 1986: 81–106.
13. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Ed., 1993.
14. Richeldi, M. and Rossotto, M.: Class-Driven Statistical Discretization of Continuous Attributes (Extended Abstract). ECML 1995: 335–338
15. Skowron, A. and Rauszer, C.: The Discernibility Matrices and Functions in Information Systems. ICS PAS Report 1/91 (Technical University of Warsaw): 1–44.
16. Van de Merckt, T.: Decision Trees in Numerical Attribute Spaces. IJCAI 1993: 1016–1021.
17. Ziarko, W.: Variable Precision Rough Set Model. Journal of Computers and System Sciences (49), 1993: 39–59.