

# Marketing data classification using Johnson’s algorithm

Zia Akbar\*

**Abstract:** *A simple method of data classification using rough set theory is described, as implemented by the tools provided in Alexander Øhrn’s software application ROSETTA. Details are given of one such tool, Johnson’s algorithm. The algorithm is applied to a typical set of marketing data from satisfaction surveys.*

**Keywords:** Rough Sets.

## 1 Introduction

Z. Pawlak’s Rough Set Theory [6] provides a popular method for data analysis and classification. The popularity of the method derives from the fact that unlike many modeling methods used in data-mining, (such as regression and discriminant analysis), rough set theory methods are *non-invasive*: they do not presume any particular model and hence are able to detect relationships in data without having to fit the latter to any such model.

Surveys investigating the satisfaction regarding the quality of a particular service (e.g. a petrol station) are important in marketing. The questions in the survey are generally structured in the following way: there are several *aspect* variables, all measuring the satisfaction of the client with respect to various aspects of the service (e.g. cleanliness of environment, efficiency and politeness of staff, reasonability of prices, etc), and one *global* satisfaction variable (“Overall, how satisfied are you with the quality of service provided?”). All the variables have the same (ordered) set of possible responses: ranging from “not satisfied” to “very satisfied”; there are usually three to five possible responses in the range.

It is of interest in marketing research to explain the global satisfaction variable using the aspect variables. Several techniques have been applied, such as tree-based models, regression analysis, and perceptual mapping techniques based on correspondence analysis.

This article describes the use of a rough set classification technique in this context, specifically using Johnson’s algorithm. It is set out as follows:

- Summary and background necessary to this article, of the rough set analysis methods developed by Pawlak et al. and explained in Komorowski [6], in particular the use of the discernibility function [13] for the extraction of prime implicants.
- An example dataset from Komorowski [6], illustrating these methods.
- An outline of how Johnson’s algorithm is used for extracting a suitable prime implicant, as implemented in Alexander Øhrn’s application ROSETTA [10].
- Application of Johnson’s algorithm to a typical marketing data set.
- (Appendix) details of a simple implementation of Johnson’s algorithm.

## 2 Background and preliminaries.

A summary is given here of the methods developed for the analysis of categorical decisional data, as developed and described in Komorowski [6].

---

\*Mail: zia@ic4life.net

The framework used here is a decisional information system,  $(U, A \cup \{d\})$ ; This can be seen as a “data table” with  $U$  as a finite set of observations, which we will call *objects*, and  $A$  as a finite set of  $m$  variables that are qualitative, i.e. can take on a finite set of values, not necessarily ordered. (If we wish to use quantitative variables, we must discretize them accordingly, by partitioning them into categories of ranges.) These variables are called *attributes*.

$d$  is a *decisional* attribute: it serves to *classify* the observations in a way which we would like to express using the other attributes. (By abuse of language we could say that  $d$  is the “dependent variable”, and  $A$  the set of “independent variables” in the model we wish to build).

We seek to extract a finite set of rules from our data table, i.e. a set of statements of the “if-then” type: “if the attributes in  $A$  take on such-and-such values, then  $d$  will take on such-and-such value”. We may then apply this set of rules to try and predict the classification of other target sets of objects with respect to the same set of attributes. This is a typical step in the data-mining process.

According to the methods of rough set theory as described by Komorowski, Pawlak, et al [6], we partition the data set using *indiscernibility classes*. To do this, we define a indiscernibility relation as follows; For a set  $B \subseteq A$ , we define the *indiscernibility relation*  $I_B$ :

$$\forall x, x' \in U, x I_B x' \iff \forall b \in B, b(x) = b(x').$$

$I_B$  is an equivalence relation, and so the classes it induces are equivalence classes, forming a partition (non-overlapping division) of the set  $U$ . Any two objects that are in a given class are indiscernible with respect to the attributes in  $B$ .

An attribute  $b \in B$  is said to be *dispensable* in  $B$  if  $I_B = I_{(B \setminus \{b\})}$ , i.e. if  $b$  can be removed from  $B$  without affecting the partition. If not,  $b$  is said to be *indispensable* in  $B$ . If all the attributes in  $B$  are indispensable in  $B$ , and  $I_B = I_A$ , then  $B$  is said to be a *reduct* for  $A$ . In other words, a reduct is a minimal set of attributes that preserves the classification of the objects.

Note: we assume that our decisional information system is *consistent*, i.e. that any two objects that are indiscernible with respect to the set of attributes  $A$  will automatically share the same value for  $d$ . This is, of course, not necessarily the case in practice, but such an inconsistent system can always be transformed to a consistent one as a preprocessing step before further analysis, by computing a *generalized decision attribute*: for each indiscernibility class, we simply take the range of decisions possible for its members, and group this range into one single “generalized” decision class. The resulting information system is then consistent with respect to this generalized decision attribute.

Skowron and Rauszer [13] give a straightforward method of computing reducts, described as follows. First construct a *discernibility matrix* from the data set. This is a table showing the differences between each pair of objects, with respect to attributes. It is therefore a  $n \times n$  matrix,  $n$  being the number of objects (individuals) in the dataset. The matrix is symmetric and has a “zero” ( $\emptyset$ ) diagonal.

The general  $(i, j)^{th}$  term  $c_{ij}$  of this matrix is the set of attributes for which the  $i^{th}$  and  $j^{th}$  object of the dataset take different values:

$$c_{ij} = \{a \in A : a(x_i) \neq a(x_j)\}$$

Alongside, construct a boolean *discernibility function*,  $f$ , which is parallel to the discernibility matrix:

$$f(\bar{a}_1, \dots, \bar{a}_m) = \bigvee \{ \bigwedge \bar{c}_{ij} : 1 \leq j < i \leq n, c_{ij} \neq \emptyset \}$$

where  $\bar{a}_1, \dots, \bar{a}_m$  are boolean variables corresponding to the  $m$  attributes in  $A$ , and similarly,  $\bar{c}_{ij} = \{\bar{a} : a \in c_{ij}\}$  is the boolean term corresponding to  $c_{ij}$ .

The set of prime implicants for the boolean function  $f$  can then be computed. This set determines all reducts (5) of  $A$ .

In our case, since we are interested in classifying our data with respect to the decision attribute  $d$ , we need reducts that influence  $d$ . From now on we will therefore deal with the case of computing *decision-relative* reducts, which is as follows (Skowron [13]):

We construct a *decision-relative discernibility matrix* from the data set. Similar to the discernibility matrix above, this table shows the differences between each pair of objects, with respect to attributes *and decision classes*. It is also a  $n \times n$  symmetric zero-diagonal matrix. The general  $(i, j)^{th}$  term  $c'_{ij}$  of this matrix is the set of attributes for which the  $i$ 'th and  $j$ 'th object of the dataset take different values, *and* are also in different decision classes:

$$c'_{ij} = \begin{cases} \{a \in A : a(x_i) \neq a(x_j)\} & \text{if } d(x_i) \neq d(x_j), \\ \emptyset & \text{if } d(x_i) = d(x_j). \end{cases}$$

From this decision-relative discernibility matrix we construct the *decision-relative discernibility function*, in exactly the same way as the discernibility function above:

$$f'(\bar{a}_1, \dots, \bar{a}_m) = \bigvee \{ \bigwedge \bar{c}'_{ij} : 1 \leq j < i \leq n, c'_{ij} \neq \emptyset \}$$

Note : the set of decision rules arrived at by the process described above may not be necessarily be the “best” ones to apply to unseen objects. We do not treat this topic in detail here, but remark that it is possible to improve the selection by choosing *dynamic reducts*, i.e. reducts which are stable with respect to random resampling of the original data table, as described by Bazan [2].

### 3 Example.

The classical example from Komorowski et al. ([6], [15]) is given here in order to illustrate the technique described above. The decisional information system is shown below. There are four attributes that are non-decisional: Degree (d), Experience (e), French (f) and Reference (r), along with a fifth “accept/reject” decisional attribute. Eight objects are listed:

	d	e	f	r	Decision
$x_1$	MBA	Medium	Yes	Excellent	Accept
$x_2$	MBA	Low	Yes	Neutral	Reject
$x_3$	MCE	Low	Yes	Good	Reject
$x_4$	MSc	High	Yes	Neutral	Accept
$x_5$	MSc	Medium	Yes	Neutral	Reject
$x_6$	MSc	High	Yes	Excellent	Accept
$x_7$	MBA	High	No	Good	Accept
$x_8$	MCE	Low	No	Excellent	Reject

We can then compute the decision-relative discernibility matrix. For example, the term  $c'_{12}$  compares the objects  $x_1$  and  $x_2$ . The two objects differ with respect to their decisions, so  $c'_{12}$  will not necessarily be  $\emptyset$ . Checking the data table, we see that the objects also differ with respect to the attributes “e” and “r”, so  $c'_{12} = \{e, r\}$ .

This results in the following decision-relative discernibility matrix:

	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	$x_8$
$x_1$	$\emptyset$							
$x_2$	$\{e,r\}$	$\emptyset$						
$x_3$	$\{d,e,r\}$	$\emptyset$	$\emptyset$					
$x_4$	$\emptyset$	$\{d,e\}$	$\{d,e,r\}$	$\emptyset$				
$x_5$	$\{d,r\}$	$\emptyset$	$\emptyset$	$\{e\}$	$\emptyset$			
$x_6$	$\emptyset$	$\{d,e,r\}$	$\{d,e,r\}$	$\emptyset$	$\{e,r\}$	$\emptyset$		
$x_7$	$\emptyset$	$\{e,f,r\}$	$\{d,e,f\}$	$\emptyset$	$\{d,e,f,r\}$	$\emptyset$	$\emptyset$	
$x_8$	$\{d,e,f\}$	$\emptyset$	$\emptyset$	$\{d,e,f,r\}$	$\emptyset$	$\{d,e,f\}$	$\{d,e,r\}$	$\emptyset$

We now compute the decision-relative discernibility function  $f'$ . Without confusing matters, we use the same letter to denote an attribute and its corresponding boolean variable, but with different typefaces. Thus, for example, the attribute ‘‘Degree’’ is written ‘‘d’’ and its boolean counterpart ‘‘ $d$ ’’.

We then simply write each discernibility matrix term as a disjunction, of the corresponding boolean variables; e.g. for  $c'_{12} = \{e,r\}$ , the corresponding term in the discernibility function is  $e \vee r$ .  $f'$  is then simply the conjunction of all the terms of the matrix given above, written as disjunctions. So, gathering all the non-null terms in the table above (starting from the top left), we find:

$$f' = (e \vee r) \wedge (d \vee e \vee r) \wedge (d \vee e) \wedge (d \vee e \vee r) \wedge (d \vee r) \wedge (e) \wedge (d \vee e \vee r) \wedge (d \vee e \vee r) \wedge (e \vee r) \wedge (e \vee f \vee r) \wedge (d \vee e \vee f) \wedge (d \vee e \vee f \vee r) \wedge (d \vee e \vee f) \wedge (d \vee e \vee f \vee r) \wedge (d \vee e \vee f) \wedge (d \vee e \vee r).$$

This is simplified using rules of boolean algebra, giving the result:

$$f' = (e \vee r) \wedge (d \vee e).$$

$(e \vee r)$  and  $(d \vee e)$  are the prime implicants of the function. Hence the decision-relative reducts concerned are  $\{e, r\}$  and  $\{d, e\}$ .

The simplification of the discernibility function using rules of boolean algebra is referred to as the *exhaustive method* of reduction. It is a procedure that is easy enough in principle but is of NP-complexity. Hence, for large datasets, it is naturally costly in terms of computer resources to compute the complete set of reducts.

In this case we may decide to accept the finding of only a subcollection of reducts instead of the whole set; i.e. only a subcollection of prime implicants of the discernibility function. This permits the use of heuristic approximation algorithms that are faster than the exhaustive method, without losing out much on effectiveness.

## 4 The use of Johnson’s algorithm.

Johnson’s reduction algorithm is an example of a heuristic algorithm of the type described above. The version referred to here is Alexander Øhrn’s adaptation of Johnson’s approximation algorithm for a minimal set cover, as implemented in his software application ROSETTA (Rough Set Toolkit For Data Analysis): this application also features the exhaustive method and several other heuristic algorithms for computing prime implicants. Johnson’s algorithm is a typical ‘‘greedy’’ algorithm with a natural tendency to find a single prime implicant of minimal length (Øhrn [10]). It provides a reduction method that is fast and efficient with the minimum of overhead; good results seem to be obtained for small training datasets; larger training sets generally lead to more rules being generated, and less decidability in classification.

A description of the algorithm is given below, as implemented in ROSETTA(Øhrn [10]):

Let  $\mathbb{S}$  be a collection of sets corresponding to the discernibility function  $f'$  computed beforehand. For instance, in our example above,  $\mathbb{S} = \{(e, r), (d, e, r) \dots\}$ , simply listing the terms we gathered above to build  $f'$ .

To find a reduct  $B$  from  $\mathbb{S}$ , the steps of the algorithm are as follows:

1. Let  $B = \emptyset$ .

2. Let  $a$  be the attribute most frequently found in the sets of  $\mathbb{S}$ . In case of ties, choose one tied attribute arbitrarily.

(In fact, we choose the attribute  $a$  that maximizes

$$\sum_{S \in \mathbb{S}, a \in S} w(S),$$

where  $w$  is a *weight* or *cost* function over the sets  $S$  in  $\mathbb{S}$ . The simplest case, which we use here, is to fix  $w=1$ .)

3. Add  $a$  to  $B$ .
4. Remove all the sets from  $\mathbb{S}$  that contain  $a$ .
5. If  $\mathbb{S} = \emptyset$  then return  $B$  as result. Else go back to step 2.

In our example above,  $e$  is the most frequent attribute, so we choose it, and delete all the terms that contain it. Then we are left with just the term  $d \vee r$ , so  $d$  and  $r$  tie for the most frequent attribute. Arbitrarily, we choose  $d$  (in our implementation, we choose the first attribute in our list of ties). No further terms are left, and the algorithm stops. So our reduct is  $\{d, e\}$ .

## 5 Extraction of rules and classification of unseen data.

Once the reduct has been found, the rules are generated simply by taking the values of the attributes for each object, “plugging” them into the reduct, and attaching the resulting statement as an implicant to the corresponding decision variable value, i.e. “gluing” them together in the form “if (statement) then (decision)” (Øhrn [10]).

In our example, we apply the object  $x_1$  to the reduct  $\{d, e\}$ . We know that for  $x_1$ ,  $d = \text{“MBA”}$  and  $e = \text{“Medium”}$ , while the value of the decision is “Accept”. So, this gives us the rule:

if ( $d = \text{“MBA”}$ ) and ( $e = \text{“Medium”}$ ) then decision = “Accept”.

We are now in a position to *classify* any previously unseen objects by using a voting strategy with the rule; an example of this is the *standard voting* strategy implemented by Øhrn [10] in ROSETTA:

1. We find all the rules whose antecedents (“conditions”) are consistent with the values taken by the object (*firing rules*).
2. For each possible decision value for the object, we calculate a number of votes for each rule corresponding to this decision value. In the “*simple count*” case, this number of votes can be a single vote, whereas in the “*support count*” case, it is taken as a number equal to the corresponding *support* value (the number of objects corresponding to the rule in the base values).
3. Next, for each decision value, we find the *certainty factor*: we divide the number of votes by a “total” (this is known as *normalisation*). For the “*simple count*” case, we can take the number of rules concerned to be either all the rules that were generated (“*All*”), or only those that are consistent with the object (“*Firing*”). For the “*support count*” case, we take the total to be the sum of the supports of either all the rules, or of just the “*firing*” rules, respectively.
4. We choose the decision value as being that for which the certainty factor is highest.

Afterwards, we can cross-tabulate the predicted decisions by the calculated decisions, in a *confusion matrix* in order to compare the performance of the algorithm.

## 6 A test on satisfaction survey data.

It was decided to test the rough set classification technique to a typical example of this type of marketing data, i.e. with the global satisfaction variable taken as the decision attribute, explained by the other aspects. The test was carried out using the ROSETTA software application, with Johnson's algorithm used as the reducer. The 5 variables had each a range of 5 possible responses. A set of anonymized data was provided from an actual product testing survey. It contained 208 objects, of which 15 were chosen as a training group to generate the decision reducts and rules, which were then applied to the remaining 193 objects (the test group).

The confusion matrix representing a typical test is shown below (predicted values in columns and actual values in rows):

	1	2	3	4	5	undefined
1	3	0	0	0	0	1
2	3	22	6	0	0	5
3	0	5	30	8	0	17
4	0	3	5	53	4	11
5	0	2	0	3	10	2
undefined	0	0	0	0	0	0

Remarks :

1. The data table was complete: there were no missing values. (ROSETTA is, however, capable of completing data tables using means, or other interpolation values, if necessary.)
2. The training group was deliberately kept small; larger training groups tended to produce a reduct consisting of all the attributes and generate less definite classification rules, thus causing more objects to be classed as "undefined".
3. The 15 members of the training group were chosen at random; however, it was ensured that each decision value was represented equally often in the sample, e.g. 3 objects were chosen at random from those having a decision value of 1, another 3 from those of decision 2, etc. This avoided under-representation of any decision class, which would cause most test objects in such a class to remain unclassified.
4. The algorithm performed reasonably well, with over 60 % of objects being classified correctly. It is interesting to note that even for incorrectly classified objects, the predicted decision class was generally adjacent to the correct decision class, as can be seen in the above table. The classes being ordinal, even incorrect predictions could be read as being "not far wrong", enabling reasonable marketing decisions. The application of this type of rough set classification to satisfaction surveys therefore seems promising.

The same set of data was also tested using a genetic algorithm that is available through ROSETTA. The results were as follows:

	1	2	3	4	5	undefined
1	3	0	0	0	0	1
2	3	22	8	0	0	4
3	0	6	37	8	0	9
4	0	3	6	53	4	10
5	0	2	0	3	10	2
undefined	0	0	0	0	0	0

The genetic algorithm produces slightly better results than Johnson's algorithm, but is *much* slower, and the overhead may not be feasible for large datasets. In general it seems that Johnson's algorithm is able to quickly generate a reasonably accurate rule, provided it is trained on small datasets.

## 7 Appendix : Implementation of Johnson's algorithm

The basic minimal covering algorithm is described below, as given in Mielikäinen [7], which quotes from Johnson [8]:

Let  $\mathcal{U}$  be a "universal" set of  $n$  elements,  $\mathbb{S} = \{S_1, S_2, \dots, S_k\}$  a collection of subsets of  $\mathcal{U}$  forming a cover for it, and  $c : \mathbb{S} \rightarrow \mathbb{Q}^+$  a "cost" function. Johnson's approximation algorithm finds a sub-collection of  $\mathbb{S}$  covering all the elements of  $\mathcal{U}$  at minimal cost.

The steps in the algorithm are as follows:

1. Let  $C = \emptyset, \mathcal{T} = \emptyset$ .
2. while  $C \neq \mathcal{U}$  do
  - Find  $S \in \mathbb{S}$  such that  $c(S) \setminus |S \setminus C|$  is minimum.
  - $\forall x \in s$ , define  $cost(x) = c(S) \setminus |S \setminus C|$ .
  - $C \leftarrow C \cup S$ .
  - $\mathcal{T} \leftarrow C \cup \{S\}$ .
3. Result =  $\mathcal{T}$ .

A simple skeleton procedure is given below in Object Pascal, which constructs a boolean discernibility function from a previously filled-in data table array, applies Johnson's algorithm to produce a reduct, then generates rules from this reduct. I developed this code using Øhrn's (1) description of the ROSETTA version as a starting point. As such, this code is intended to be clear in principle rather than computationally optimized. It can easily be converted to other object-oriented languages such as Java and C++.

```
//=====
// Procedure implementing Johnson's algorithm
// Zia Akbar, 2002
//=====

procedure JohnsonReducer;
var // ===== variables used =====
  i,j,k: integer;
  s: string;
  nv : integer; //number of non-decisional attributes
  //the decisional variable is the (nv+1)th variable

  DataTab : array of array of strings;
  //DataTab is a 2D array representing the data table,
  //presumed to ALREADY contain all the data of n objects and nv+1 variables.

  //TIntegerList is a list type inheriting from the
  //basic list object TList; for which the objects
  //contained in the list are integers.
  //A TIntegerList can be sorted in increasing order.

  til: TIntegerList; //dummy lists
  tsl : TStringList;

  tslDiscFn: TStringList;
  //Each string in the list tslDiscFn is a term of the
  //discernibility function, WITHOUT repetition.
  //Literals are separated from one another
  //by asterisks. The non-decisional attributes are
  //indicated as numbers from 1 to nv.
```

```

tilAttrib      : TIntegerList;
//list of integers representing attributes

tilWeightvalue : TIntegerList;
//list of corresponding weight function values
idx   : integer;
AttribMax : integer;
//attribute for which the weight function is maximum
wmax   : double;

const WeightFunctionValue = 1;

begin //start of main procedure
//Build up list representing the discernibility function
for i := 0 to n-2 do
  for j := i+1 to n-1 do
    begin
      //if the decision is not the same
      //for the i'th and j'th objects then...
      if DataTable[nv,i]<>DataTable[nv,j] then
        begin
          {...build the term}
          til := TIntegerList.Create; //list of integers
                                     //representing the term

          for k := 0 to nv-1 do
            begin
              //if the value of the attribute is not
              //the same for both objects, then
              //add the attribute to the term
              if DataTable[k,i]<>DataTable[k,j] then til.add(k+1);
            end;
          til.Sort; //sort the term
          s := ''; //s = string representing the term
          for k := 0 to til.Count-1 do
            s := s + IntToStr(til[k])+'*';

            // Now that the term has been built, we add it to the list,
            // but only if it isn't there already.
            // This avoids redundancy, and is an application
            // of the simplifying rule:
            // "A intersection A = A" for a given term A.}
            if tslDiscFn.IndexOf(s)=-1
              then tslDiscFn.AddObject(s,til)
              else til.free;
          end;
        end;
      end;
    end;
  end;
end;

```

```

//===== Reduction by Johnson's algorithm =====
//tilReduct = list of integers representing our reduct, initially empty.

while (tslDiscFn.count>0) do //while the discernibility function is nonempty,
begin //carry out a reduction step.
  tilAttrib.Clear;
  tilWeightValue.Clear;
  //===== fill-in tilAttrib, tilWeightValue
  for i := 0 to tslDiscFn.count-1 do
    begin
      til := TIntegerList(tslDiscFn.Objects[i]);
      for j := 0 to til.Count-1 do
        begin
          idx := tilAttrib.IndexOf(til[j]);
          if idx=-1
            then
              begin
                tilAttrib.Add(til[j]);
                tilWeightValue.Add(WeightFunctionValue);
              end
            else
              begin
                tilWeightValue[idx] := tilWeightValue[idx] + WeightFunctionValue;
              end;
            end;
          end;
        end;

// ===== Find the attribute for which the weight function is maximum
AttribMax := -1;
wmax := 0;
for i := 0 to tilAttrib.Count-1 do
  begin
    if wmax<tilWeightValue[i] then
      begin
        wmax := tilWeightValue[i];
        AttribMax := tilAttrib[i];
      end;
    end;
  //Add the attribute AttribMax to our reduct...
  tilReduct.Add(AttribMax);
  //...and remove all terms from tslDiscFn that contain this attribute
  for i := tslDiscFn.count-1 downto 0 do
    begin
      til := TIntegerList(tslDiscFn.Objects[i]);
      if til.IndexOf(AttribMax)>-1 then
        begin //delete term containing attribute
          til.Free;
          tslDiscFn.delete(i);
        end;
      end;
    end;
  //Sort the reduct, which is now complete.
  tilReduct.sort;
end; //end of main procedure

```

## References

- [1] Øyvind Tuset Aasheim, Helge Grenager Solheim. *Rough sets as a framework for data-mining*. 1996.
- [2] Jan G. Bazan. *A comparison of dynamic and non-dynamic rough set methods for extracting laws from decision tables*. 1998.
- [3] Ivo Düntsch, Günther Gediga. *Rough set data analysis*. 1999.
- [4] Craig S. Holman. *Boolean expressions - an overview*. 2000.
- [5] Mathias Kegelmann. *Normal Forms and Minimization*. 2001.
- [6] Jan Komorowski, Zdzislaw Pawlak, Lech Polkowski, Andrzej Skowron. *Rough Sets: A Tutorial*. 1998.
- [7] Taneli Mielikäinen. *Approximation Algorithms: An Introduction and Some Covers*. 2002.
- [8] D. S. Johnson. *Approximation algorithms for combinatorial problems*. Journal of Computer and System Sciences, 1974.
- [9] Torulf Mollestad. *A rough set approach to data-mining: extracting a logic of default rules from data*. Doctoral thesis, 1997.
- [10] Alexander Øhrn. ROSETTA Technical Reference Manual <http://www.idi.ntnu.no/aleks/rosetta> 2001.
- [11] Alexander Øhrn *Discernibility and Rough Sets in Medicine: Tools and Applications*. Doctoral thesis, 1999.
- [12] Eric SanJuan *Mémoire Maîtrise de mathématiques discrètes*. 1994.
- [13] A. Skowron, C. Rauszer. *The discernibility matrices and functions in information systems*. 1992.
- [14] C. Umans. *Hardness of approximating  $\Sigma_2^p$  problems*. 1999.
- [15] Website on Rough Set Theory. <http://www.kbs.twi.tudelft.nl/Education/Cyberles/Trondheim>, 2000.

## 8 Acknowledgements

Sincere and heartfelt thanks to *Eric SanJuan*, for his immense encouragement.