How to: Simplify decision tables using Rough Sets. The rs package for R.

Alber Sanchez.
Institute für Geoinformatik.
Westfälische Wilhelms Universität Münster.
Weseler Strasse 253, 48151 Münster, Deutschland.
alber.sanchez@uni-muenster.de

December 8, 2013

Abstract

This document is a brief introductory guide to the use of the R package "rs". The package allows users to reduce knowledge by simplifying rules in decision tables according to Rough Set theory. It is not a complete implementation of Rough Set theory; it just has the minimum functions to simplify decision tables.

Contents

1 Introduction

The "rs" package pretends to be an R implementation of the section 6.3 "Simplification of Decision Tables" of the book written by [**?**]. In the context of artificial intelligence, Rough Set theory is considered a solution alternative to the classificatory problem, allowing to discard superfluous or irrelevant information to focus in the most determinant conditions for taking a decision. Rough Set theory is known for being able to deal with contradictory or even incomplete information, making no assumptions about the internal structure of the data. Rough Sets theory is unable to deal with continuous variables which is a clear disadvantage. For more details about Rough Set theory consult [**?**].

Outline First, a revision of the classes of the package is made (sections 1 to 6) and at the end, examples of its use are given (section 7).

2 Rule

A rule is a set of conditions which imply a decision. An example rule goes like:

$$C1 \wedge C2 \wedge C3 \Rightarrow D$$

Where C1, C2, C3 are conditions and D is the decision taken if conditions are met.

The following example is a free interpretation of two decision tables found in [**?**], for that reason the conclusions derived from them probably would not make sense but serve for the goal of showing the use of the package. Assume that the decision of a student of going to class depends on the following conditions:

- Weather (condition 1).
  1. Cold.
  2. Foggy.
- Class time (condition 2).
  1. Morning.
  2. Noon.
- Class difficulty (condition 3).
  1. Hard.
  2. Too hard.
- Distance to school (condition 4).
  1. Close.
  2. Not so close.
- Decision.
  1. Not go.
  2. Go.

Assume also that the option "Not sure" is represented by 0 for all conditions and decision.

For the purpose of this package, a rule could be represented in R by a numeric vector, where the last element is the decision taken and the remaining elements are conditions values.

The rule *if weather is cold* $\wedge$ *I'm not sure when the class is* $\wedge$ *the class is too hard* $\wedge$ *the class is not so close* $\Rightarrow$ *I'm not sure of going to class* Could be represented in R using the vector:

```
> rule <- c(1,1,1,1,0)
> print(rule)

[1] 1 1 1 1 0
```

3 Decision table

A decision table is a set of rules with the same set of conditions. Imagine a couple of expert students who were questioned about when to go or not to class, according to the conditions defined above. The result is an expert system of rules which can be coded as an R matrix where each row is a rule. This matrix can be used for creating an object of the class Decision Table, like this:

```
> exampleMatrix1 <- matrix(c(1,0,2,2,0,
+ 0,1,1,1,2,
+ 2,0,0,1,1,
+ 1,1,0,2,2,
+ 1,0,2,0,1,
+ 2,2,0,1,1,
+ 2,1,1,1,2,
+ 0,1,1,0,1),ncol = 5,byrow=TRUE)
> # Decision table creation
> dt1 <- new(Class="DecisionTable",decisionTable = exampleMatrix1)
> # Decision table creation alterative
> dt1 <- decisionTable(exampleMatrix1)
```

Due to the intrinsic nature of information, a decision table probably contains duplicated or even contradictory rules. A contradictory or inconsistent rule is a rule in a decision table for which there is at least another rule in the same decision table with exactly the same conditions and a different decision. If any pair of rules have one single different condition they are consistent no matter the decision and if a pair of rules in a decision table have the exactly the same conditions and decision, in other words if they are a duplicated rule, they are consistent.

The "rs" package has some tools for identifying inconsistent rules; for example if the last column of the example matrix 1 is removed, then some rules become inconsistent:

```
> dt <- decisionTable(exampleMatrix1[,-5])
> print(dt)

*** Class DecisionTable, method Print ***
   C1 C2 C3 D
R1 1  0  2  2
R2 0  1  1  1
R3 2  0  0  1
R4 1  1  0  2
R5 1  0  2  0
R6 2  2  0  1
R7 2  1  1  1
R8 0  1  1  0
******* End Print (DecisionTable) *******

> computeConsistencyMatrix(dt)

      R1    R2   R3   R4   R5   R6   R7   R8
R1 FALSE    NA   NA   NA   NA   NA   NA   NA
R2  TRUE FALSE   NA   NA   NA   NA   NA   NA
R3  TRUE  TRUE TRUE   NA   NA   NA   NA   NA
R4  TRUE  TRUE TRUE TRUE   NA   NA   NA   NA
R5 FALSE  TRUE TRUE TRUE TRUE   NA   NA   NA
R6  TRUE  TRUE TRUE TRUE TRUE TRUE   NA   NA
R7  TRUE  TRUE TRUE TRUE TRUE TRUE TRUE   NA
R8  TRUE FALSE TRUE TRUE TRUE TRUE TRUE TRUE
```

Checking the consistency matrix by columns, it can be seen that rules R1 and R5 are inconsistent between them, just like R2 and R8. A summary of the consistency matrix can be obtained by:

```
> checkConsistency(dt)

   R1    R2    R3    R4    R5    R6    R7    R8
FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE
```

But now it is not possible to identify which rule subset is inconsistent.

4 Discernibility Matrix

A discernibility matrix identifies the differences in condition values for each pair of rules in a decision table. The discernibility matrix of a Decision Table object can be obtained this way:

```
> print(dt1)

*** Class DecisionTable, method Print ***
   C1 C2 C3 C4 D
R1 1  0  2  2  0
R2 0  1  1  1  2
R3 2  0  0  1  1
R4 1  1  0  2  2
R5 1  0  2  0  1
R6 2  2  0  1  1
R7 2  1  1  1  2
R8 0  1  1  0  1
******* End Print (DecisionTable) *******

> computeDiscernibilityMatrix(dt1)

*** Class DiscernibilityMatrix, method Show ***
* DiscernibilityMatrix (limited to a matrix 10x10) =
   R1          R2          R3          R4          R5          R6
R1          NA          NA          NA          NA          NA          NA
R2 C1,C2,C3,C4          NA          NA          NA          NA          NA
R3    C1,C3,C4    C1,C2,C3          NA          NA          NA          NA
R4       C2,C3    C1,C3,C4    C1,C2,C4          NA          NA          NA
R5          C4 C1,C2,C3,C4    C1,C3,C4    C2,C3,C4          NA          NA
R6 C1,C2,C3,C4    C1,C2,C3          C2    C1,C2,C4 C1,C2,C3,C4          NA
R7 C1,C2,C3,C4          C1       C2,C3    C1,C3,C4 C1,C2,C3,C4       C2,C3
R8 C1,C2,C3,C4          C4 C1,C2,C3,C4    C1,C3,C4    C1,C2,C3 C1,C2,C3,C4
   R7          R8
R1          NA          NA
R2          NA          NA
R3          NA          NA
R4          NA          NA
R5          NA          NA
R6          NA          NA
```

```
R7          NA          NA
R8       C1,C4          NA
******* End Show (DiscernibilityMatrix) *******
```

Here it is evident that R1 and R2 have different values for conditions 1, 2, 3 and 4 but R1 and R5 only difference is at condition 4. Matrix elements where there just a one condition difference allow identifying the condition core of the decision table, in this example the core is made of C1, C2 and C4. The importance of the core is that allows finding the condition reducts.

5 Condition Reduct

A condition reduct is a decision table where the superfluous conditions have been removed. The "superfluousness" of a condition is determined by comparing the indiscernibility function of the decision table and the indiscernibility function of the decision table without the superfluous condition. In [?] there is no reference to such a thing as a "condition" or "value" reduct, these are just artifacts created for the "rs" package.

In a Decision Table object can exist simultaneously many different condition reducts but since the goal here is the reduction of knowledge, the smallest condition reduct are the central target. By the smallest condition reduct is understood the reduct with the minimum number of conditions but sometimes there can be more than one condition reduct meeting this requirement; in that case, the set of smallest condition reduct is called a family. The Decision Table class has 3 methods to find condition reducts.

The first method just returns a Condition Reduct object of the first found smallest condition reduct in a decision table. That condition reduct could belong to a family or be the only one but it is not possible to know that using this method:

```
> firstCR <- findFirstConditionReduct(dt1)
> print(firstCR)

*** Class ConditionReduct, method Print ***
   C1 C2 C3 C4 D
R1 1  0  NA 2  0
R2 0  1  NA 1  2
R3 2  0  NA 1  1
R4 1  1  NA 2  2
R5 1  0  NA 0  1
R6 2  2  NA 1  1
R7 2  1  NA 1  2
R8 0  1  NA 0  1
******* End Print (ConditionReduct) *******
```

Here, it can be seen that C3 is superfluous for making the decision D. The second method finds the smallest family of Condition Reduct objects which is returned as a list:

```
> smallestFamilyCR <- findSmallestReductFamilyFromCore(dt1)
```

In this case the core is a condition reduct itself, in such situations the smallest condition reduct family is composed of one condition reduct which is the core. Finally, there is a method to find all the reducts:

```
> allCR <- findAllReductsFromCore(dt1)
```

Note that in this case, there are just 2 condition reducts, the core and the original decision table itself.

6 Value Reduct

A value reduct is a condition reduct where the superfluous conditions of each rule have been removed. A Value Reduct object can be obtained from a Condition Reduct object like this:

```
> vr <- computeValueReduct(firstCR)
> print(vr)
```

```
*** Class ValueReduct, method Print ***
   C1 C2 C3 C4 D
R1 NA 0  NA 2  0
R2 0  NA NA 1  2
R2 NA 1  NA 1  2
R3 2  0  NA NA 1
R3 NA 0  NA 1  1
R4 1  1  NA NA 2
R4 NA 1  NA 2  2
R5 NA NA NA 0  1
R6 NA 2  NA NA 1
R7 2  1  NA NA 2
R7 NA 1  NA 1  2
R8 NA NA NA 0  1
******* End Print (ValueReduct) *******
```

Note the additional NA values on each rule. They mean the value for that condition is not needed for making a decision.

A Value Reduct object can calculate metrics of the rule:

```
> computeSupportConsistency(vr,dt1)
```

```
   C1 C2 C3 C4 D consistentCount inconsistentCount support consistency
R1 NA  0 NA  2 0               1                 0   0.125           1
R2  0 NA NA  1 2               1                 0   0.125           1
R2 NA  1 NA  1 2               2                 0   0.250           1
R3  2  0 NA NA 1               1                 0   0.125           1
R3 NA  0 NA  1 1               1                 0   0.125           1
R4  1  1 NA NA 2               1                 0   0.125           1
R4 NA  1 NA  2 2               1                 0   0.125           1
R5 NA NA NA  0 1               2                 0   0.250           1
R6 NA  2 NA NA 1               1                 0   0.125           1
R7  2  1 NA NA 2               1                 0   0.125           1
```

```
R7 NA  1 NA  1 2              2                0   0.250          1
R8 NA NA NA  0 1              2                0   0.250          1
```

The "consistentCount" and "inconsistentCount" is the number of times the rule is consistent in the Decision Table object, together they are the total number of times the rule appears. Support is the ratio between total number of times the rule appears and the total number of rules in the decision table. Consistency is the ration between "consistentCount" and the total number of times the rule appears. In other words, support measures the classificatory capacity of the rule in a decision table and consistency is a measure of roughness of the decision of the rule, which is useful in case of presence of inconsistent rules.

The rules in a Value Reduct object could be used to classify a Decision Table object returning a new Decision Table object with the decisions from the value reduct. For example, the result of applying the Value Reduct object to the Decision Table object from which it was originated is:

```
> classDT <- classifyDecisionTable(vr,dt1)
> print(classDT)

*** Class DecisionTable, method Print ***
    C1 C2 C3 C4 D
R1  1  0  2  2  0
R2  0  1  1  1  2
R3  0  1  1  1  2
R4  2  1  1  1  2
R5  2  0  0  1  1
R6  2  0  0  1  1
R7  1  1  0  2  2
R8  1  1  0  2  2
R9  1  0  2  0  1
R10 0  1  1  0  1
R11 2  2  0  1  1
R12 2  1  1  1  2
R13 0  1  1  1  2
R14 2  1  1  1  2
R15 1  0  2  0  1
R16 0  1  1  0  1
******* End Print (DecisionTable) *******
```

Note that the classified Decision Table object has more rules than the original.

7 Examples

7.1 Example 1

Assume there is a rule set which has been broken in two, one for training and the other for classifying, named respectively "exampleMatrix1" and "exampleMatrix2". In this example any solution is acceptable.

```
> exampleMatrix1 <- matrix(c(1,0,0,1,1,
+ 1,0,0,0,1,
+ 0,0,0,0,0,
+ 1,1,0,1,0,
+ 1,1,0,2,2,
+ 2,1,0,2,2,
+ 2,2,2,2,2),ncol = 5,byrow=TRUE)
> exampleMatrix2 <- matrix(c(1,0,2,2,0,
+ 0,1,1,1,2,
+ 2,0,0,1,1,
+ 1,1,0,2,2,
+ 1,0,2,0,1,
+ 2,2,0,1,1,
+ 2,1,1,1,2,
+ 0,1,1,0,1),ncol = 5,byrow=TRUE)
```

Next, two Decision Table objects are build, one for each rule set:

```
> dt1 <- decisionTable(exampleMatrix1)
> dt2 <- decisionTable(exampleMatrix2)
```

The object dt1 could be explored using the methods showed so far or even the duplicated (using "removeDuplicatedRulesDT") rules can be removed to improve the computation performance, but there are no duplicated rules in dt1. The decisions of the rules in dt2 are not important since the classification process will change their values.

Now, a condition and value reduct are obtained from the classificatory decision table:

```
> cr1 <- findFirstConditionReduct(dt1)
> vr1 <- computeValueReduct(cr1)
```

The condition core is calculated internally by "findFirstConditionReduct". Again, the duplicated rules could be removed from the Condition Reduct objet (using "removeDuplicatedRulesCR"), but there is none in this example, but the case of the Value Reduct object is different, so:

```
> vr1 <- removeDuplicatedRulesVR(vr1)
```

It is the turn of evaluating the rules obtained using the original Decision Table including the duplicated rules if any. Note that this evaluation could be carried using the other Decision Table object, dt2.

```
> computeSupportConsistency(vr1,dt1)
```

```
   C1 C2 C3 C4 D consistentCount inconsistentCount   support consistency
R1  1  0 NA NA 1               2                 0 0.2857143           1
R1 NA  0 NA  1 1               1                 0 0.1428571           1
R2  1 NA NA  0 1               1                 0 0.1428571           1
R3  0 NA NA NA 0               1                 0 0.1428571           1
R4 NA  1 NA  1 0               1                 0 0.1428571           1
R5 NA NA NA  2 2               3                 0 0.4285714           1
R6  2 NA NA NA 2               2                 0 0.2857143           1
R7 NA  2 NA NA 2               1                 0 0.1428571           1
```

At this point, some important conclusions about the rules in the Decision Table object can be drawn. Taking as starting point what was stated above (in section 2), it is possible to say the difficulty of the class is irrelevant for students when deciding going to class because condition 3 is not part of this reduct. Rule 3 states that if the student has no clue about the weather, no matter the remaining conditions, he is not sure of going. Rules 5 and 8 states that no matter what, if the classroom is far they go to class, this makes no sense but remember the example data is a free interpretation of decision tables found in [**?**].

Assuming the results are satisfactory, now it is time of classifying the other part of the rules. Remember that the decisions of dt2 are overwritten with the decision from vr1:

```
> dt3 <- classifyDecisionTable(vr1,dt2)
> print(dt3)

*** Class DecisionTable, method Print ***
    C1 C2 C3 C4 D
R1  1  0  2  2  1
R2  1  0  2  0  1
R3  2  0  0  1  1
R4  1  0  2  0  1
R5  0  1  1  1  0
R6  0  1  1  0  0
R7  0  1  1  1  0
R8  2  1  1  1  0
R9  1  0  2  2  2
R10 1  1  0  2  2
R11 2  0  0  1  2
R12 2  2  0  1  2
R13 2  1  1  1  2
R14 2  2  0  1  2
******* End Print (DecisionTable) *******
```

It can be seen that more than one rule from vr1 applied to the rules in dt2 and for that reason dt3 has more rules than dt2.

7.2 Example 2

This example wants to reproduce the results of the example given in [**?**] in section 6.3, page 72.

Decision Table object creation:

```
> dtMatrix <- matrix(c(1,0,0,1,1,
+ 1,0,0,0,1,
+ 0,0,0,0,0,
+ 1,1,0,1,0,
+ 1,1,0,2,2,
+ 2,1,0,2,2,
+ 2,2,2,2,2),ncol = 5,byrow=TRUE)
> dt <- decisionTable(dtMatrix)
```

Condition reduct:

```
> cr <- findFirstConditionReduct(dt)
> print(cr)

*** Class ConditionReduct, method Print ***
   C1 C2 C3 C4 D
R1 1  0  NA 1  1
R2 1  0  NA 0  1
R3 0  0  NA 0  0
R4 1  1  NA 1  0
R5 1  1  NA 2  2
R6 2  1  NA 2  2
R7 2  2  NA 2  2
******* End Print (ConditionReduct) *******
```

Value reduct:

```
> vr <- computeValueReduct(cr)
> print(vr)

*** Class ValueReduct, method Print ***
   C1 C2 C3 C4 D
R1 1  0  NA NA 1
R1 NA 0  NA 1  1
R2 1  0  NA NA 1
R2 1  NA NA 0  1
R3 0  NA NA NA 0
R4 NA 1  NA 1  0
R5 NA NA NA 2  2
R6 2  NA NA NA 2
R6 NA NA NA 2  2
R7 2  NA NA NA 2
R7 NA 2  NA NA 2
R7 NA NA NA 2  2
******* End Print (ValueReduct) *******
```

Notice how some rule ids are duplicated (on the left side), indicating the origin rule of the Condition Reduct object for the rule in the Value Reduct object. This Value Reduct has the same results showed in the table 7 of [**?**] at page 75.

The rest of the example is about making combinations of rules (2 rules come from R1, 2 from R2, 2 from R6, 3 from R7 and 1 from each of the rest, that is 24 combinations) in order to find the smallest possible value reduct. So far, the "rs" package does not include a method for making all the possible combinations and it must be taken into account that this criteria does not rely on metrics like support and consistency, it just seeks the smallest value reduct.

Using a combination of R functions and the "rs" package the example continues building 2 options of the value reduct:

```
> vrMat <- getValueReduct(vr)# Matrix representation of the value reduct
> vr1 <- valueReduct(cr,vrMat[c(1,4,5,6,7,9,10),])# Pick rules by matrix row index
> vr2 <- valueReduct(cr,vrMat[c(1,3,5,6,7,9,12),])
```

The object vr1 corresponds to table 8 and vr2 to table 9 on page 76 of [**?**]. Now duplicated rules from vr2 are removed:

```
> vr3 <- removeDuplicatedRulesVR(vr2)
```

Object vr3 corresponds to table 10 on page 77 of [**?**]. Finally, rules are re-numbered

```
> vr3Mat <-  getValueReduct(vr3)
> rownames(vr3Mat) <- paste("R",1:nrow(vr3Mat),sep="")
> vr4 <- valueReduct(cr,vr3Mat)
> print(vr4)

*** Class ValueReduct, method Print ***
   C1 C2 C3 C4 D
R1 1  0  NA NA 1
R2 0  NA NA NA 0
R3 NA 1  NA 1  0
R4 NA NA NA 2  2
******* End Print (ValueReduct) *******
```

The Value reduct object vr4 corresponds to table 11 on page 77 of [**?**].

This example exposes the ability of the "rs" package to reproduce the section "6.3 Simplification of Decision Tables" of [**?**].

References

[Pawlak] Pawlak, Zdzislaw 1991 *Rough Sets: Theoretical Aspects of Reasoning About Data* Dordrecht: Kluwer Academic Publishing.