Introducing NRough Framework

Sebastian Widz[✉]

Systems Research Institute, Polish Academy of Sciences,
ul. Newelska 6, 01-447 Warsaw, Poland
widz@nrough.net

Abstract. In this article we present the new machine learning framework called *NRough*. It is focused on rough set based algorithms for feature selection and classification i.e. computation of various types of decision reducts, bireducts, decision reduct ensembles and rough set inspired decision rule induction. Moreover, the framework contains other routines and algorithms for supervised and unsupervised learning. *NRough* is written in C# and compliant with .NET Common Language Specification (CLS). Its architecture allows easy extendability and integration.

Keywords: Rough sets · Approximate decision reducts · Bireducts · .NET · C#

1 Introduction

Interest in machine learning tools and algorithms has been huge in recent years and is still growing. There is a wide range of applications on the market that use various machine learning routines. There is however still only a few solutions compatible with the Microsoft .NET framework that can provide machine learning algorithms. Those that exist are rather focused on numerical rather than symbolic methods and so far none of these has included rough set [1] based algorithms.

Machine learning models that are based on mathematically sophisticated methods may achieve high accuracy but they are hardly understandable by users who expect not only accurate results but also easy yet meaningful explanation how these results were obtained. Models relaying on symbolic, e.g., rule based methods may be less accurate but more intuitive and understandable for humans [2]. In both cases, feature subset selection leads to an increase of interpretability and practical usefulness of machine learning models.

Symbolic methods focus on finding relationships in data, typically reported in a form of rules in a feature-value language. The rules are built with a use of basic logical operators. Examples include the rule induction methods such as learning if-then rules [3] or decision trees [4].

Rough sets have proven to be a successful tool in feature selection (see e.g. [5]). The rough set approach is based on *decision reducts* – irreducible subsets of features, which determine specified decision classes in (almost) the same

L. Polkowski et al. (Eds.): IJCRS 2017, Part I, LNAI 10313, pp. 669–689, 2017.
DOI: 10.1007/978-3-319-60837-2_53

degree as the original set of features. Determining decisions can be interpreted analogously to, e.g., functional or multi-valued dependencies in relational databases. Subsets of features providing exactly the same degree of determination as the original set are often referred as *crisp* decision reducts, in opposite to *approximate* decision reducts [6] where some controlled decrease of determination is allowed. By specifying a threshold for allowed decrease of determination, one can address the balance between decision model's simplicity and accuracy. Indeed, it is easier to search for smaller subsets of features yielding simpler sub-models under loosened constraints for decision determination, although too weak constraints may also cause poorer accuracy. However, even relatively less accurate sub-models may lead towards very accurate final model, if the processes of sub-models' design are appropriately synchronized.

NRough is a set of libraries written in C# programming language focusing on rough sets and other symbolic machine learning methods. It contains a number of algorithms for searching approximate decision reducts and constructing decision models. All presented algorithms has been successfully used in our previous research and proven their value. The framework is aimed to be used by researchers who can extend it and test their methods against already implemented models. The second user group are developers and system integrators who can include described routines in their applications.

NRough can be downloaded from *GitHub* Repository [7] as well as from its dedicated website [8] in a form of a Microsoft Visual Studio solution containing source code for all described libraries. The sources include unit test code that presents use case examples as well as unit testing procedures.

We present the framework's key features as well as formal definitions behind implemented algorithms in Sect. 2. We describe data representation, approximate decision reducts and decision reduct classifier ensembles in this section. Moreover, we list other supervised and unsupervised machine learning algorithms included in *NRough*. Finally, we list included features related to model evaluation. Next, in Sect. 3 we describe the architecture of our solution. We put licensing information in Sect. 4. In Sect. 5 we describe other rough set based frameworks. The last Sect. 6 concludes this paper and includes draft of a road map as well as a direction in which we would like the framework to evolve.

2 Key Features

NRough framework contains a number of algorithms for searching approximate decision reducts and constructing decision models based on the rough set theory. Moreover, we added a number of decision rule based classifiers known from machine learning. Last but not least, the framework contains routines for classifier validation and results presentation. Below, we present the list of implemented features starting with data representation description.

2.1 Data Representation

We follow data representation in a form of a decision table which is a well known structure in the rough sets domain. *Decision table* is a tuple $\mathbb{A} = (U, A \cup \{d\})$, where U is a finite set of objects, A is a finite set of attributes and $d \notin A$ is a distinguished decision attribute. We refer to elements of U using their ordinal numbers $i = 0, ..., |U| - 1$ as well as by unique record identifier (if such exists in the data set). We treat attributes $a \in A$ as functions $a : U \to V_a$, where V_a denotes the set of values of a. Values $v_d \in V_d$ correspond to decision classes that we want to describe using the values of attributes in A. The framework internally encodes attribute values using *signed long* base type which allows generic approach for data access and avoiding boxing/unboxing from origin types i.e. faster computations. Internal values can be however converted to their original typed values using a dictionary lookup methods. There are no restrictions about input types, that are automatically recognized during data loading.

Decision tables can be loaded from text files (including comma delimited files and RSES 1.0 format [9]) as well as from the `System.Data.DataTable` instance which is often used in .NET to store SQL query results. The framework includes a number of filters to manipulate the data such as removal of selected attributes or records based on a given user criteria. Filter concept is also used to define more sophisticated data manipulations such as numeric value discretization.

One of the key concepts in rough sets theory is the definition of indiscernibility relation. For any subset of attributes $B \subseteq A$ and the universe of objects $x \in U$ we are able to define an information vector $B(x) = [a_{i_1}(x), \ldots, a_{i_{|B|}}(x)]$ where $a_{i_j}(x)$ are values of attributes $a_{i_j} \in B$ and $j = 1, \ldots, |B|$. We can also denote the set of all *B-information* vectors, which will then occur in $\mathbb{A}$, as $V_B = \{B(x) : x \in U\}$. Each subset $B \subseteq A$ partitions the space U onto so called *equivalence classes* that can be enumerated as $v_1, \ldots, v_{|V_B|}$. For such division we get the partition space denoted as $U/B = \{E_1, \ldots, E_t\}$ where $E_t \subseteq U$ for $t = 1, \ldots, |V_B|$. Each equivalence class is defined as $E_t = \{x \in U : B(x) = v_t\}$. *NRough* utilizes this concept in a form of a dedicated data structure which is used in many scenarios like calculating functions *information entropy*, *majority* or *relative gain* functions to name a few. The *majority* function is used in many ways by framework algorithms e.g. for approximate decision reducts computation as well as for decision tree pre-pruning or branch split calculation.

Last but not least, the library contains a number of benchmark data sets taken from UCI repository [10]. These data can be accessed with a predefined methods loading the data into memory including their meta data.

2.2 Approximate Decision Reducts

Attribute selection plays an important role in knowledge discovery. It establishes the basis for more efficient classification, prediction and approximation models. Attribute selection methods originating from the theory of rough sets aim at searching for so called decision reducts – irreducible subsets of attributes that satisfy predefined criteria for keeping *enough* information about decision classes. *NRough* contains a number of algorithms for computing *approximate* decision reducts (as well as *crisp* decision reducts when approximation threshold is set to 0). All reduct computation algorithms are based on heuristic approach and many utilize parallel computing.

We define an (F, ε)-approximate decision reduct [11] where F is a measure $F(d|\cdot) : 2^{|A|} \rightarrow \Re$ which evaluates the degree of influence $F(d|B)$ of subset $B \subseteq A$ in d. Below we present the definition as well as the general routine for computing (F, ε)-approximate decision reducts as Algorithm 1 called (F, ε)-REDORD [11]).

Definition 1. *Let $\varepsilon \in [0, 1)$ and $\mathbb{A} = (U, A \cup \{d\})$ be given. We say that $B \subseteq A$ is an (F, ε)-approximate decision reduct, iff it is an irreducible subset of attributes satisfying the following condition:*

$$F(B) \geq (1 - \varepsilon)F(A) \tag{1}$$

Algorithm 1. Modified (F, ε)-REDORD using *Reach* and *Reduce* operations

Input: $\varepsilon \in [0, 1)$, $\mathbb{A} = (U, A \cup \{d\})$, $\sigma : \{1, ..., n\} \rightarrow \{1, ..., n\}$, $n = |A|$
Output: $B \subseteq A$

```
 1: $B \leftarrow \emptyset$
 2: for $i = 1 \rightarrow n$ do        //Reach
 3:     if $F(B \cup \{a_{\sigma(i)}\}) < (1 - \varepsilon)F(A)$ then
 4:         $B \leftarrow B \cup \{a_{\sigma(i)}\}$
 5:     else
 6:         break
 7:     end if
 8: end for
 9: for $j = |B| \rightarrow 1$ do        //Reduce
10:     if $F(B \setminus \{a_{\sigma(i)}\}) \geq (1 - \varepsilon)F(A)$ then
11:         $B \leftarrow B \setminus \{a_{\sigma(i)}\}$
12:     end if
13: end for
14: return $B$
```

The framework defines three types of F measures: $\gamma(B)$ [1] which is based on so called *positive region*, Majority $M(B)$ [11] and Relative Gain $R(B)$ [12]. Other user defined measures can be used with (F, ε)-approximate reduct computation algorithm.

$$M(B) = \frac{1}{|U|} \sum_{E \in U/B} \max_{k \in V_d} |X_k \cap E| \tag{2}$$

$$R(B) = \frac{1}{|V_d|} \sum_{E \in U/B} \max_{X \in U/\{d\}} \frac{|X \cap E|}{|X|} \tag{3}$$

$$\gamma(B) = \frac{1}{|U|}|POS(B)| = \frac{1}{|U|} \sum_{E \in U/B : P(X|E)=1} |E| \tag{4}$$

In [13] it was shown how to compute approximate decision reducts over a universe of weighted objects and that two different weighting schemes lead to an unified way of computing $M(B)$ and $R(B)$ measures that is for $\mathbf{1} : U \rightarrow \{1\}$ we have $M_{\mathbf{1}}(B) = M(B)$ and for $r(u) = \frac{1}{|\{x \in U : d(x) = d(u)\}|}$ we obtain $M_r(B) = R(B)$.

Definition 2. *Let $\varepsilon \in [0,1)$, $\mathbb{A} = (U, A \cup \{d\})$ and $\omega : U \rightarrow [0, +\infty)$ be given. We say that $B \subseteq A$ is an (ω, ε)-approximate decision reduct, iff it is an irreducible subset of attributes satisfying the following condition:*

$$M_\omega(B) \geq (1 - \varepsilon) M_\omega(A) \tag{5}$$

$$M_\omega(B) = \frac{1}{|U|_\omega} \sum_{E \in U/B} \max_{k \in V_d} |X_k \cap E|_\omega \tag{6}$$

$$|Y|_\omega = \sum_{u \in Y} \omega(u) \tag{7}$$

Moreover the framework contains algorithms for computing decision bireducts and their derivatives γ-bireducts and relative-bireducts [14]. Below we present definitions as well as pseudo code as Algorithm 2. In [15] we showed relationships between (F, ε)-approximate decision reducts and different types of bireducts.

Definition 3. *Let $\mathbb{A} = (U, A \cup \{d\})$ be a decision system. A pair (B, X), where $B \subseteq A$ and $X \subseteq U$, is called a decision bireduct, iff B discerns all pairs $i, j \in X$ where $d(i) \neq d(j)$, and the following properties hold:*

1. *There is no $C \subsetneq B$ such that C discerns all pairs $i, j \in X$ where $d(i) \neq d(j)$;*
2. *There is no $Y \supsetneq X$ such that B discerns all pairs $i, j \in Y$ where $d(i) \neq d(j)$.*

Definition 4. *Let $\mathbb{A} = (U, A \cup \{d\})$ be a decision system. A pair (B, X), where $B \subseteq A$ and $X \subseteq U$, is called a decision γ-bireduct, iff B discerns all pairs $i \in X, j \in U$ where $d(i) \neq d(j)$, and the following properties hold:*

1. *There is no $C \subsetneq B$ such that C discerns all pairs $i \in X, j \in U$ where $d(i) \neq d(j)$;*
2. *There is no $Y \supsetneq X$ such that B discerns all pairs $i \in Y, j \in U$ where $d(i) \neq d(j)$.*

In [16] we presented the new definition of so called generalized majority decision function, which can be treated as an extension to well known generalized decision function. We also showed the definition of generalized approximate majority decision reducts. The pseudo code is presented as Algorithm 3. An interesting extension in to use so called *exceptions* which on one hand allow further feature reduction in the main model and on the other hand store details about outlayers. Both definitions are implemented in *NRough*.

Algorithm 2. Decision bireduct calculation for a decision system $\mathbb{A} = (U, A \cup \{d\})$

Input: $\mathbb{A} = (U, A \cup \{d\})$, $\sigma : \{1, ..., n+m\} \rightarrow \{1, ..., n+m\}$, $m = |A|$, $n = |U|$
Output: $(B \subseteq A, X \subseteq U)$

```
 1: $B \leftarrow A$; $X \leftarrow \emptyset$
 2: for $i = 1 \rightarrow n + m$ do
 3:     if $\sigma(i) \leq n$ then
 4:         if $B \setminus \{a_{\sigma(i)}\} \Rrightarrow_X d$ then
 5:             $B \leftarrow B \setminus \{a_{\sigma(i)}\}$
 6:         end if
 7:     else
 8:         if $B \Rrightarrow_{X \cup \{\sigma(i)-n\}} d$ then
 9:             $X \leftarrow X \cup \{\sigma(i) - n\}$
10:         end if
11:     end if
12: end for
13: return $(B, X)$
```

Definition 5. *For any decision table $\mathbb{A} = (U, A \cup \{d\})$ and approximation threshold $\varepsilon \in [0, 1)$ one can consider generalized approximate majority decision function $m_d^\varepsilon : 2^U \rightarrow 2^{V_d}$ that is taking the following form:*

$$m_d^\varepsilon(E) = \{k : |X_k \cap E| \geq (1 - \varepsilon) \max_j |X_j \cap E|\} \tag{8}$$

Definition 6. *Let $\mathbb{A} = (U, A \cup \{d\})$ be given. We say that $B \subseteq A$ is a m_d^ε-decision superreduct, if and only if the following condition holds:*

$$\bigforall_{x,y \in U} m_d^\varepsilon([x]_A) \neq m_d^\varepsilon([y]_A) \Rightarrow \bigexists_{a \in B} a(x) \neq a(y) \tag{9}$$

We say that B is a m_d^ε-decision reduct, if and only if it is a m_d^ε-superreduct and none of it proper subsets satisfy the above condition.

2.3 Approximate Decision Reduct Classifier Ensembles

Approximate decision reducts usually include less attributes than classical reducts. On the other hand, they may generate if-then rules that make mistakes even within the training samples. For noisy data sets it is to some extent desirable. Nevertheless, some methods for controlling those mistakes should be considered. For example, if the goal is to construct a classification model based on several approximate decision reducts, then – by following ideas taken from machine learning [17] – one may wish to assure that if-then rules generated by different reducts do not repeat the same mistakes on the training data. For this purpose, we can consider a mechanism aiming at diversification of importance of particular objects while searching for different approximate reducts. The same

Algorithm 3. Generalized Majority Decision Reduct

```
Input: $\mathbb{A} = (U, A \cup \{d\})$, $\varepsilon \in [0, 1)$, $\sigma : \{1, ..., n\} \to \{1, ..., n\}$, $n = |A|$
Output: $B \subseteq A$
1:  for all $E_A \in U/A$ do
2:      $d(x) \leftarrow m_d^{\varepsilon}(E_A)$
3:  end for
4:  $B \leftarrow A$
5:  for $i = 1 \to n$ do
6:      $B \leftarrow B \setminus \{a_{\sigma(i)}\}$
7:      $stop \leftarrow 0$
8:      for all $E_B \in U/B$ do
9:          for all $(x_1, x_2) \in E_B$ do
10:             if $d(x_1) \cap d(x_2) = \varnothing$ then
11:                 $stop \leftarrow 1$
12:                 break
13:             else
14:                 $d(x_1) \leftarrow d(x_1) \cap d(x_2)$
15:                 $d(x_2) \leftarrow d(x_1)$
16:             end if
17:         end for
18:         if $stop = 1$ then
19:             $B \leftarrow B \cup \{a_{\sigma(i)}\}$
20:             break
21:         end if
22:     end for
23: end for
24: return $B$
```

mechanisms are used in classifier ensemble methods. These methods perform usually better than their components used independently [18,19]. Combining classifiers is efficient especially if they are substantially different from each other. In fact, the feature subsets applied in ensembles can be relatively smaller than in case of a single feature subset approach, if we can guarantee that combination of less accurate classifier components (further referred as weak classifiers) will lead back to satisfactory level of determining decision or preserving information about decision.

NRough includes several mechanisms for approximate decision reduct classifier ensembles learning. One method is based on well known Adaptive Boosting algorithm [20]. In *NRough* we introduced an *AdaBoost* version which use decision rules derived from approximate decision reducts [21] - the pseudo code is presented in Algorithm 5.

When a reduct ensemble is used to create decision rules one can consider a weak classifier output combination method. We implemented several voting mechanisms described in [22]. We present different voting options in Table 1 in the way compliant with (ω, ε)-approximate decision reducts. The voting weights are presented in a slightly changed form where

$$X_E^\omega = \underset{X \in U/\{d\}}{\operatorname{argmax}} |X \cap E|_\omega \tag{10}$$

Algorithm 4. Generalized Majority Decision Reduct with exceptions

Input: $\mathbb{A} = (U, A \cup \{d\})$, $\phi \in [0,1)$, $\varepsilon \in [0,1)$, $\sigma : \{1, ..., n\} \rightarrow \{1, ..., n\}$, $n = |A|$
Output: $B \subseteq A$

```
 1: for all $E_A \in U/A$ do
 2:     $d(x) \leftarrow m_d^{\phi,0}(E_A)$
 3: end for
 4: $B \leftarrow A$, $c \leftarrow |U|$
 5: for $i = 1 \rightarrow n$ do
 6:     $B \leftarrow B \setminus \{a_{\sigma(i)}\}$
 7:     $stop \leftarrow 0$
 8:     Shuffle($E_B$)
 9:     for all $E_B \in U/B$ do
10:         for all $(x_1, x_2) \in E_B$ do
11:             if $d(x_1) \cap d(x_2) = \varnothing$ then
12:                 $c \leftarrow c - |E_B|$
13:                 if $c < (1 - \phi) * |U|$ then
14:                     $stop \leftarrow 1$
15:                     break
16:                 else
17:                     SaveExceptionRule($E_B$)
18:                 end if
19:             else
20:                 $d(x_1) \leftarrow d(x_2) \leftarrow d(x_1) \cap d(x_2)$
21:             end if
22:         end for
23:         if $stop = 1$ then
24:             $B \leftarrow B \cup \{a_{\sigma(i)}\}$
25:             break
26:         end if
27:     end for
28: end for
29: return $B$
```

Table 1. Six options of weighting decisions by if-then rules, corresponding to the consequent coefficient types plain, ω-confidence and ω-coverage, and antecedent coefficient types single and ω-support. $|E|_\omega$ denotes the support of a rule's left side. X_E^ω is defined by formula (10).

	Single	ω-support
Plain	1	$\|E\|_\omega / \|U\|_\omega$
ω-confidence	$\|X_E^\omega \cap E\|_\omega / \|E\|_\omega$	$\|X_E^\omega \cap E\|_\omega / \|U\|_\omega$
ω-coverage	$(\|X_E^\omega \cap E\|_\omega / \|X_E^\omega\|_\omega)/(\|E\|_\omega / \|U\|_\omega)$	$\|X_E^\omega \cap E\|_\omega / \|X_E^\omega\|_\omega$

Algorithm 5. AdaBoost with (ω, ε)-Approximate Reducts as Weak Classifier

```
Input: $\mathbb{A} = (U, A \cup \{d\})$, $n = |A|$, $\varepsilon \in [0, 1)$, integer $T$ specifying number of iterations
Output: Approximate Reduct Ensemble $S = \{r_1, ..., r_s\}$, $s \leq T$
Initialize: $\omega_i = 1/n$ for $i = 1, 2, 3, ..., n$
 1: Calculate error threshold $\epsilon_0 = 1 - M_\omega(\emptyset)$;
 2: for $t = 1 \to T$ do
 3:     Generate permutation $\sigma$
 4:     Create $(\omega, \varepsilon)$-approximate decision reduct $r_t$ based on permutation $\sigma$
 5:     Generate decision rules based on conditional attributes from reduct $r_t$
 6:     Classify training examples
 7:     Calculate the error $\epsilon_t$
 8:     if $\epsilon_t > \epsilon_0$ or $\epsilon_t = 0$ then
 9:         Break
10:     end if
11:     Calculate weak classifier confidence $\alpha_t$
12:     Update and normalize object weights $\omega$
13: end for
14: Normalize $\alpha$
15: return $S$
```

Another introduced method for decision reduct diversification is based on decision reduct hierarchical clustering where a distance between reducts is based on binary vectors created according to Formula 11. Figure 1 presents an example of a dendrogram created based on hierarchical clustering of approximate decision reducts. By choosing a cut level we create a number of reducts groups. From each group a single reduct is selected and used as a base for creating a weak classifier. Weak classifiers together form a classifier ensemble.

$$\overrightarrow{v_B}[k] = \begin{cases} 1, & \text{if} \quad d(x_k) = \underset{X \in U/\{d\}}{\operatorname{argmax}} |X \cap E| \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

The framework also includes selection mechanisms which allow to select from a reduct pool those reducts that meet a user defines criteria e.g. contain least number of features, generate least number of decision rules etc.

2.4 Other Machine Learning Algorithms

Except rough sets inspired classifiers the framework includes a number of decision rule induction algorithms. These algorithms can be combined with rough set based feature selection methods defined in the previous section. Current implementation includes the following routines:

Decision lists generation routine based on feature subsets and a given decision table.

Majority voting based on a feature subset and a given decision table [23].

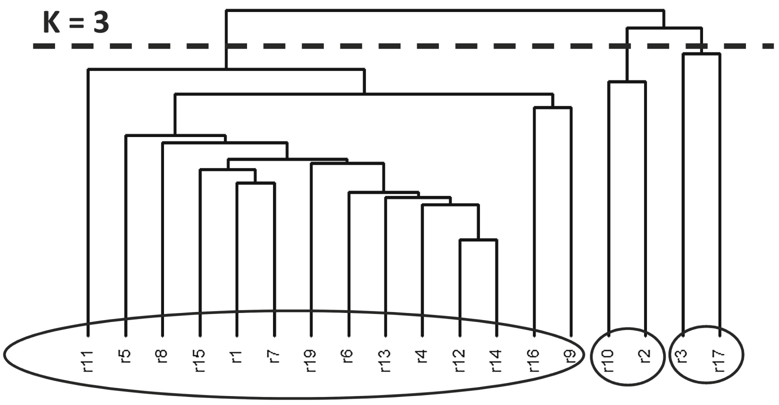

Fig. 1. Dendrogram created based on hierarchical clustering of 18 approximate decision reducts.

Decision trees (**C4.5** [24], **ID3** [25]) implementation supporting numerical and nominal attributes types. The impurity functions can be easily exchanged with one another and include *information entropy*, *gini index*, *majority* function or other user defined methods. Decision tree base class include **pruning** option - current implementation includes an *error based pruning* and a *reduced error pruning* [26]. We also introduce a new pre-pruning method based on the *majority* function.

Random forest implementation with option to select ensemble size, base decision tree type and data sampling method.

1R rule inducer which, as suggested in [27], could be used to calculate data set baseline accuracy.

Constant decision classifier which classifies all object as majority decision from training data set.

The framework includes a set of unsupervised algorithms based on **Hierarchical clustering** [28] with different *linkage* and *distance* methods. Model construction algorithms that work only with nominal data can utilize a number of numerical attribute discretization methods, both supervised and unsupervised. **Supervised** include hierarchical methods based on *information entropy* [29,30] and *majority* function. **Unsupervised** include *equal width* and *equal frequency* binning. Most of implemented algorithms can work with weighted instances.

2.5 Model Evaluation

Proper evaluation and error estimation is crucial in constructing and comparing decision models. One of the key features in *NRough* is decision model evaluation. Currently the framework provides the following evaluation methods [31]: *k-fold n-repeated cross validation* (CV), *leave-one-out* CV, *bootstrap* with *out-of-the-bag* testing and finally *n-repeated hold out*.

Each evaluation test can return detailed information about experiment results in a form of a formatted table which can be saved as a .CSV or .TEX file. Additionally, results can be presented in a graphical form using an interface to *R* environment [32] - the graphical presentation as well as the latex tabular output are still under development but the working code is available in the repository and can be customized according to your need. The *classification result* class interface contains information such as *error* and *accuracy* rates, *balanced accuracy* (useful for testing imbalanced data sets), *error standard deviation*, *confidence*, *coverage*, *f-score*, *recall* and *precision* as well as classification *confusion table*. If model definition allows it, there is possibility to include information about complexity e.g. size of a decision tree, number of rules, average length of rules, etc.

3 Architecture

NRough is a Microsoft .NET based framework written in C# programming language. The source code is CLS compliant which enables to use it in other .NET languages. Currently the framework is provided as a set of libraries targeting .NET 4.6.1 and 64 bit architecture.

The libraries have the following structure:

NRough.Core Contains generic data structures and extensions methods.

NRough.Data Responsible for data handling. It defines the *decision table* interface and *equivalence class* collection as well as routines for providing meta data and interface to data filtering.

NRough.MachineLearning Contains approximate decision reduct computation algorithms as well as other described machine learning models and routines.

NRough.Math Contains special functions used across other modules e.g. statistical functions or distance metrics used in clustering.

NRough.Tests.* Contains a set of test fixtures which except unit testing purpose serve as a code sample repository. Each of the above listed libraries has its own unit test library e.g. `NRough.Data` is tested by `NRough.Tests.Data`.

The framework is currently dependent on the following external libraries (except standard .NET): `Math.NET.Numerics` [33], `NUnit` [34] and `R.NET` [35].

4 License

NRough libraries are provided under *GNU Lesser General Public License ver. 3* (GNU LGPLv3). This means that provided source can be used for research, commercial and non-commercial purposes without any charges as long as GNU LGPLv3 restrictions are satisfied. Copyright and license notices must be preserved. Contributors provide an express grant of patent rights. However, a larger work using the licensed work through interfaces provided by the licensed work may be distributed under different terms and without source code for the larger work. The source code is provided "as is" without warranty of any kind, express or implied. In no event shall the authors or copyright holders be liable for any claim, damages or other liability. Complete license can be found in [36].

5 Other Frameworks

Let us focus on other rough set related frameworks developed by various researches. First of all, let us mention LERS - a system for Learning from Examples based on Rough Sets [37]. LERS contained two rules' induction algorithms (LEM1 and LEM2) that could cope with inconsistent data. LERS contained also a number of algorithms for handling missing data and numerical attribute discretization. Its performance was comparable with AQ15 and C4.5 algorithms.

Secondly let us mention more complete GUI-based systems: Rough Set Exploration System (RSES) [9] and ROSETTA [38] (a toolkit for analyzing tabular data within a framework of rough sets). Both solutions shared the same computational kernel developed at the Group of Logic, Institute of Mathematics, University of Warsaw, Poland and finally ROSE2 (Rough Sets Data Explorer) [39] which is a software implementing basic elements of the rough set theory and rule discovery techniques. It was created at the Laboratory of Intelligent Decision Support Systems of the Institute of Computing Science in Poznan, Poland. RSES, ROSETTA as well as ROSE2 contained many different algorithms ranging from data preprocessing, filtering, discretization, rule induction, to classification and feature selection.

All mentioned above solutions are according to authors knowledge no longer maintained. These solutions were based on Java or C++ and its source code was not open. More recently, there were several attempts to revive RSES in a form of an open source Java based library distributed under GNU GPL license called *RSESLib* [40].

Last but not least we need to mention *RoughSets* [41] and *RapidRoughSets* [42] packages as a most recent development in rough sets domain. The former package contains the core computational methods for rough and fuzzy sets based on already mentioned *R* statistical software. The latter package is the GUI extension based on *RapidMiner* software.

In .NET domain there were so far, according to authors knowledge, no attempts to publish a machine learning framework containing rough set based algorithms. We would like to mention Microsoft Azure Machine Learning [43] which is a cloud based service integrating many different solutions resulting in a platform worth considering for system integrators. Its is however a commercial solution. We also would like to mention Accord.NET framework [44] which is an open source machine learning framework focusing on numerical methods as well as on machine vision. This framework could complement some general machine learning routines not yet implemented in *NRough*.

6 Conclusions and Future Work

We have created *NRough* framework based on our experience and research focused on approximate decision reducts done over past few years. In the beginning presented methods had been developed separately, but recently the whole source code went through major refactoring process resulting in presented solution. We have added other machine learning routines to the framework for two reasons: First to combine well known proven machine learning algorithms with rough sets, secondly, in order to be able to compare their performance against rough set inspired classifiers.

The whole framework was so far developed by a single person and there is still much to be done. First of all the framework needs strong API documentation and more examples. We are planning to add this in the nearest future and publish it on-line Secondly, we would like to add more rough set related algorithms in order to create a comprehensive library of different decision reduct computation routines. Thirdly, we are planning to extend the approximate decision reduct selection and diversification criteria. Last but not least, there are some development tasks to complete like graphical presentation methods using *R* interface. Moreover the framework is currently targeting .NET version 4.6.1 which allows to compile it only on Windows platform. We plan to extend its compatibility *.NET Core* to be able to use it on *Linux* and *OSX* platforms, but so far .NET Core lacks some important functionality so we are waiting for its new releases.

Appendix NRough Code Samples

Sample 1: 10-fold cross validation of C4.5 decision tree

```
//load data
var data = DecisionTable.Load("data.txt", FileFormat.CSV);

//create 10-fold 25-repeated cross validation
var cv = new CrossValidation(data, 10, 25);

//create C4.5 decision tree and run cv evaluation
var c45 = new DecisionTreeC45();
var result = cv.Run<DecisionTreeC45>(c45);

//output result
Console.WriteLine("Train Error: {0}", result.Error);
```

Sample 2: Random forest based on C4.5 decision trees

```
//load data from a CSV file
var data = DecisionTable.Load("german.data", FileFormat.CSV);
DecisionTable train, test;
var splitter = new DataSplitterRatio(data, 0.8);
splitter.Split(out train, out test);
//Initialize and Learn Random Forest
var forest = new DecisionForestRandom<DecisionTreeC45>();
```

```
forest.Size = 500;
forest.Learn(train, train
    .SelectAttributeIds(a => a.IsStandard).ToArray());
//Validate on test data set
var result = Classifier.Default.Classify(forest, test);
//Output the results
Console.WriteLine(result);
```

Sample 3: Generate (F, ε)-approximate decision reducts using reduct factory

```
//load data
var data = Data.Benchmark.Factory.Golf();
//set parameters for reduct factory
var parm = new Args();
parm.SetParameter(ReductFactoryOptions.DecisionTable, data);
parm.SetParameter(ReductFactoryOptions.ReductType,
    ReductTypes.ApproximateDecisionReduct);
parm.SetParameter(ReductFactoryOptions.FMeasure,
    (FMeasure) FMeasures.Majority);
parm.SetParameter(ReductFactoryOptions.Epsilon, 0.05);
//compute reducts
var reducts =
    ReductFactory.GetReductGenerator(parm).GetReducts();
//output reducts and attributes
foreach (IReduct reduct in reducts)
    Console.WriteLine(reduct.Attributes.ToArray().ToStr());
```

Sample 4: Generate (ω, ε)-decision reducts using reduct factory

```
//load benchmark data
var data = Data.Benchmark.Factory.Zoo();

//set object weights using r(u) weighting scheme
data.SetWeights(new WeightGeneratorRelative(data).Weights);

//split data into training and testing sets
DecisionTable train, test;
var splitter = new DataSplitterRatio(data, 0.8);
splitter.Split(out train, out test);

//set parameters for reduct factory
var parm = new Args();
parm.SetParameter(ReductFactoryOptions.DecisionTable, train);
parm.SetParameter(ReductFactoryOptions.ReductType,
    ReductTypes.ApproximateDecisionReduct);
parm.SetParameter(ReductFactoryOptions.FMeasure,
    (FMeasure)FMeasures.MajorityWeighted);
parm.SetParameter(ReductFactoryOptions.Epsilon, 0.05);

//compute reducts
var reductGenerator = ReductFactory.GetReductGenerator(parm);
var reducts = reductGenerator.GetReducts();
```

```

//select 10 reducts with least number of attributes
var bestReduct = reducts
    .OrderBy(r => r.Attributes.Count).Take(10);

//create decision rules based on reducts
var decisionRules = new ReductDecisionRules(bestReducts);

//when test instance is not recognized
//set output as unclassified
decisionRules.DefaultOutput = null;

//classify test data
var result = Classifier.DefaultClassifer
    .Classify(decisionRules, test);

//output accuracy and coverage
Console.WriteLine("Accuracy: {0}", result.Accuracy);
```

Sample 3: Boosting (ω, ε)-decision reduct based classifier

```
//load training and testing DNA (spieces) data sets
var train = Data.Benchmark.Factory.Dna();
var test = Data.Benchmark.Factory.DnaTest();

//set weights
var weightGen = new WeightGeneratorConstant(train,
    1.0 / (double)train.NumberOfRecords);
train.SetWeights(weightGen.Weights);

//create parameters for reduct factory
var parm = new Args();
parm.SetParameter(ReductFactoryOptions.ReductType,
    ReductTypes.ApproximateDecisionReduct);
parm.SetParameter(ReductFactoryOptions.FMeasure,
    (FMeasure)FMeasures.MajorityWeighted);
parm.SetParameter(ReductFactoryOptions.Epsilon, 0.05);
parm.SetParameter(ReductFactoryOptions.NumberOfReducts, 100);
parm.SetParameter(ReductFactoryOptions.ReductComparer,
    ReductRuleNumberComparer.Default);
parm.SetParameter(ReductFactoryOptions.SelectTopReducts, 1);

//create weak classifier prototype
var prototype = new ReductDecisionRules();
prototype.ReductGeneratorArgs = parm;

//create ada boost ensemble
var adaBoost = new AdaBoost<ReductDecisionRules>(prototype);
adaBoost.Learn(train,
    train.SelectAttributeIds(a => a.IsStandard).ToArray());

```

```
//classify test data set
var result = Classifier.Default.Classify(adaBoost, test);

//print result header & result
Console.WriteLine(ClassificationResult.TableHeader());
Console.WriteLine(result);
```

Sample 4: (F, ε)-decision reduct ensemble using hierarchical clustering diversification

```
//load training and testing DNA (spieces) data sets
var train = Data.Benchmark.Factory.Dna();
var test = Data.Benchmark.Factory.DnaTest();

//create reduct diversification
var reductDiversifier
    = new HierarchicalClusterReductDiversifier();
reductDiversifier.Data = train;
reductDiversifier.Distance = ReductDistance.Hamming;
reductDiversifier.Linkage = ClusteringLinkage.Average;
reductDiversifier.NumberOfReducts = 10;

//create parameters for reduct factory
var parm = new Args();
parm.SetParameter(ReductFactoryOptions.ReductType,
    ReductTypes.ApproximateDecisionReduct);
parm.SetParameter(ReductFactoryOptions.FMeasure,
    (FMeasure)FMeasures.MajorityWeighted);
parm.SetParameter(ReductFactoryOptions.Epsilon, 0.05);
parm.SetParameter(ReductFactoryOptions.NumberOfReducts, 100);
parm.SetParameter(ReductFactoryOptions.Diversify,
    reductDiversifier);

var rules = new ReductDecisionRules();
rules.ReductGeneratorArgs = parm;
rules.DecisionIdentificationMethod
    = RuleQualityMethods.Confidence;
rules.RuleVotingMethod = RuleQualityMethods.Coverage;

//classify test data set and show results
var result = Classifier.Default.Classify(rules, test);
Console.WriteLine(result);
```

Sample 5: Generate bireducts using class hierarchy

```
//load training data set
var train = Data.Benchmark.Factory.Dna();

//generate 100 permutations based on attributes and objects
var permGenerator =
    new PermutationGeneratorAttributeObject(train, 0.5);
var permutations = permGenerator.Generate(100);

```

```
//setup gamma-bireduct generator
//generate bireducts based on permutations
var bireductGammaGenerator = new BireductGammaGenerator();
bireductGammaGenerator.DecisionTable = train;
bireductGammaGenerator.Permutations = permutations;
var bireducts = bireductGammaGenerator.GetReducts();

//for each bireduct show its attributes and supported objects
foreach (var bireduct in bireducts)
{
    Console.WriteLine(
        bireduct.Attributes.ToArray().ToStr());

    Console.WriteLine(
        bireduct.SupportedObjects.ToArray().ToStr());
}
```

Sample 8: Compute Generalized Majority Decision Reducts

```
//load training data set
var train = Data.Benchmark.Factory.Dna();

//setup reduct factory parameters
Args parms = new Args();
parms.SetParameter(ReductFactoryOptions.DecisionTable, train);
parms.SetParameter(ReductFactoryOptions.ReductType,
    ReductTypes.GeneralizedMajorityDecision);
parms.SetParameter(ReductFactoryOptions.WeightGenerator,
    new WeightGeneratorMajority(train));
parms.SetParameter(ReductFactoryOptions.Epsilon, 0.05);
parms.SetParameter(ReductFactoryOptions.PermutationCollection,
    new PermutationCollection(10,
    train.SelectAttributeIds(a => a.IsStandard)
        .ToArray()));

//generate reducts
var reductGenerator = ReductFactory.GetReductGenerator(parms);
var reducts = reductGenerator.GetReducts();
```

Sample 9: Compute Generalized Majority Decision Reducts with exceptions

```
//load training and test data sets
var train = Data.Benchmark.Factory.Dna();
var test = Data.Benchmark.Factory.DnaTest();

//setup reduct factory parameters
Args parms = new Args();
parms.SetParameter(ReductFactoryOptions.DecisionTable, train);
parms.SetParameter(ReductFactoryOptions.ReductType,
    ReductTypes.GeneralizedMajorityDecision);
parms.SetParameter(ReductFactoryOptions.WeightGenerator,
    new WeightGeneratorMajority(train));
```

```
parms.SetParameter(ReductFactoryOptions.Epsilon, 0.05);
parms.SetParameter(ReductFactoryOptions.PermutationCollection,
    new PermutationCollection(10,
        train.SelectAttributeIds(a => a.IsStandard)
            .ToArray()));
parms.SetParameter(ReductFactoryOptions.UseExceptionRules,
    true);

//generate reducts with exceptions
var reductGenerator = ReductFactory.GetReductGenerator(parms);
var reducts = reductGenerator.GetReducts();

foreach (var reduct in reducts) {
    var r = reduct as ReductWithExceptions;
    foreach (var exception in r.Exceptions) {
        Console.WriteLine(exception.Attributes
            .ToArray().ToStr());
        Console.WriteLine(exception.SupportedObjects
            .ToArray().ToStr());
    }
}

var rules = new ReductDecisionRules(reducts);
rules.DecisionIdentificationMethod
    = RuleQualityMethods.Confidence;
rules.RuleVotingMethod = RuleQualityMethods.SingleVote;
rules.Learn(train, null);

//classify test data set
var result = Classifier.Default.Classify(rules, test);

//show results
Console.WriteLine(result);
```

Sample 10: Decision table discretization

```
var data = Data.Benchmark.Factory.Vehicle();

DecisionTable train, test;
var splitter = new DataSplitterRatio(data, 0.8);
splitter.Split(out train, out test);

var tableDiscretizer = new TableDiscretizer(
    new IDiscretizer[]
    {
        //try to discretize using Fayyad MDL Criterion
        new DiscretizeFayyad(),

        //in case Fayyad MDL is to strict
        //use standard entropy and 5 buckets
        new DiscretizeEntropy(5)
});

tableDiscretizer.FieldsToDiscretize = train
    .SelectAttributeIds(a => a.IsStandard && a.CanDiscretize());

var filter = new DiscretizeFilter();
filter.TableDiscretizer = tableDiscretizer;
filter.Compute(train);

foreach(int attributeId in tableDiscretizer.FieldsToDiscretize)
{
    var fieldDiscretizer = filter
        .GetAttributeDiscretizer(attributeId);

    Console.WriteLine("Attribute {0} was discretized with {1}",
        attributeId, fieldDiscretizer.GetType().Name);
    Console.WriteLine("Computed Cuts: {0}",
        fieldDiscretizer.Cuts.ToStr());
}

var trainDisc = filter.Apply(train);
var testDisc = filter.Apply(test);
```

References

1. Pawlak, Z., Skowron, A.: Rudiments of rough sets. Inf. Sci. **177**(1), 3–27 (2007)
2. Widz, S., Ślęzak, D.: Rough set based decision support - models easy to interpret. In: Peters, G., Lingras, P., Ślęzak, D., Yao, Y. (eds.) Rough Sets: Selected Methods and Applications in Management and Engineering, pp. 95–112. Springer, Heidelberg (2012)
3. Agrawal, R., Imielinski, T., Swami, A.N.: Mining association rules between sets of items in large databases. In: Buneman, P., Jajodia, S. (eds.) Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., 26–28 May 1993, pp. 207–216. ACM Press (1993)
4. Rokach, L., Maimon, O.: Data Mining with Decision Trees: Theory and Applications. World Scientific, Singapore (2014)
5. Świniarski, R.W., Skowron, A.: Rough set methods in feature selection and recognition. Pattern Recogn. Lett. **24**(6), 833–849 (2003)
6. Nguyen, H.S., Ślęzak, D.: Approximate reducts and association rules - correspondence and complexity results. In: Zhong, N., Skowron, A., Ohsuga, S. (eds.) RSFDGrC 1999. LNCS, vol. 1711, pp. 137–145. Springer, Heidelberg (1999). doi:10.1007/978-3-540-48061-7_18
7. Widz, S.: NRough framework git repository (2017). https://www.github.org/nrough/
8. Widz, S.: NRough framework website (2017). http://www.nrough.net
9. Bazan, J.G., Szczuka, M.S.: The rough set exploration system. In: Peters, J.F., Skowron, A. (eds.) Transactions on Rough Sets III. LNCS, vol. 3400, pp. 37–56. Springer, Heidelberg (2005)

10. Lichman, M.: UCI machine learning repository (2013)
11. Ślęzak, D.: Rough sets and functional dependencies in data: foundations of association reducts. In: Gavrilova, M.L., Tan, C.J.K., Wang, Y., Chan, K.C.C. (eds.) Transactions on Computational Science V. LNCS, vol. 5540, pp. 182–205. Springer, Heidelberg (2009). doi:10.1007/978-3-642-02097-1_10
12. Ślęzak, D., Ziarko, W.: The investigation of the Bayesian rough set model. Int. J. Approximate Reason. **40**(1–2), 81–91 (2005)
13. Widz, S., Ślęzak, D.: Attribute subset quality functions over a universe of weighted objects. In: Kryszkiewicz, M., Cornelis, C., Ciucci, D., Medina-Moreno, J., Motoda, H., Raś, Z.W. (eds.) RSEISP 2014. LNCS, vol. 8537, pp. 99–110. Springer, Cham (2014). doi:10.1007/978-3-319-08729-0_9
14. Stawicki, S., Ślęzak, D., Janusz, A., Widz, S.: Decision bireducts and decision reducts - a comparison. Int. J. Approximate Reason. **84**, 75–109 (2017)
15. Stawicki, S., Widz, S.: Decision bireducts and approximate decision reducts: comparison of two approaches to attribute subset ensemble construction. In: 2012 Federated Conference on Computer Science and Information Systems (FedCSIS), pp. 331–338. IEEE (2012)
16. Widz, S., Stawicki, S.: Generalized majority decision reducts. In: 2016 Federated Conference on Computer Science and Information Systems (FedCSIS), pp. 165–174. IEEE (2016)
17. Kuncheva, L.I., Diez, J.J.R., Plumpton, C.O., Linden, D.E.J., Johnston, S.J.: Random subspace ensembles for fMRI classification. IEEE Trans. Med. Imaging **29**(2), 531–542 (2010)
18. Breiman, L.: Bagging predictors. Mach. Learn. **24**(2), 123–140 (1996)
19. Kuncheva, L.I.: Combining Pattern Classifiers: Methods and Algorithms. Wiley, Hoboken (2014)
20. Freund, Y., Schapire, R.E.: Experiments with a new boosting algorithm. In: Saitta, L. (ed.) ICML, pp. 148–156. Morgan Kaufmann, Burlington (1996)
21. Widz, S.: Boosting approximate reducts. In: Techniki informacyjne: teoria i zastosowania: wybrane problemy, Instytut Badań Systemowych PAN, vol. 5, no. 17, pp. 129–148 (2015)
22. Ślęzak, D., Widz, S.: Is it important which rough-set-based classifier extraction and voting criteria are applied together? In: Szczuka, M., Kryszkiewicz, M., Ramanna, S., Jensen, R., Hu, Q. (eds.) RSCTC 2010. LNCS, vol. 6086, pp. 187–196. Springer, Heidelberg (2010). doi:10.1007/978-3-642-13529-3_21
23. Kohavi, R.: The power of decision tables. In: Lavrac, N., Wrobel, S. (eds.) ECML 1995. LNCS, vol. 912, pp. 174–189. Springer, Heidelberg (1995). doi:10.1007/3-540-59286-5_57
24. Quinlan, J.R.: C4.5: Programs for Machine Learning. Elsevier, Amsterdam (2014)
25. Quinlan, J.R.: Induction of decision trees. Mach. Learn. **1**(1), 81–106 (1986)
26. Quinlan, J.R.: Simplifying decision trees. Int. J. Man-Mach. Stud. **27**(3), 221–234 (1987)
27. Holte, R.C.: Very simple classification rules perform well on most commonly used datasets. Mach. Learn. **11**(1), 63–90 (1993)
28. Gurrutxaga, I., Arbelaitz, O., Martín, J.I., Muguerza, J., Pérez, J.M., Perona, I.: SIHC: a stable incremental hierarchical clustering algorithm. In: ICEIS, vol. 2, pp. 300–304 (2009)
29. Fayyad, U.M., Irani, K.B.: Multi-interval discretization of continuous-valued attributes for classification learning. In: Thirteenth International Joint Conference on Artificial Intelligence, vol. 2, pp. 1022–1027. Morgan Kaufmann Publishers (1993)

30. Kononenko, I.: On biases in estimating multi-valued attributes. In: 14th International Joint Conference on Articial Intelligence, pp. 1034–1040 (1995)
31. Friedman, J., Hastie, T., Tibshirani, R.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer Series in Statistics. Springer, Heidelberg (2009)
32. R Development Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2008). ISBN 3-900051-07-0
33. Rüegg, C., Marcus, C.: Math.NET numerics (2017). https://numerics.mathdotnet.com/. Accessed 11 Feb 2017
34. Poole, C., Prouse, R., Busoli, S., Colvin, N.: NUnit framework (2017) https://www.nunit.org/. Accessed 11 Feb 2017
35. Perraud, J.M.: R.NET github repository (2017). https://github.com/jmp.75/rdotnet. Accessed 11 Feb 2017
36. Free Software Foundation: Gnu lesser general public license. https://www.gnu.org/licenses/lgpl-3.0.en.html. Accessed 11 Feb 2017
37. Grzymala-Busse, J.W.: LERS-a data mining system. In: Maimon, O., Rokach, L. (eds.) Data Mining and Knowledge Discovery Handbook, pp. 1347–1351. Springer, Heidelberg (2005)
38. Komorowski, J., Øhrn, A., Skowron, A.: Case studies: public domain, multiple mining tasks systems: ROSETTA rough sets. In: Handbook of Data Mining and Knowledge Discovery, pp. 554–559. Oxford University Press, Inc. (2002)
39. Predki, B., Słowiński, R., Stefanowski, J., Susmaga, R., Wilk, S.: ROSE - software implementation of the rough set theory. In: Polkowski, L., Skowron, A. (eds.) RSCTC 1998. LNCS, vol. 1424, pp. 605–608. Springer, Heidelberg (1998). doi:10.1007/3-540-69115-4_85
40. Wojna, A.: RSESLib. (2017). http://rseslib.mimuw.edu.pl/. Accessed 11 Feb 2017
41. Riza, L.S., Janusz, A., Bergmeir, C., Cornelis, C., Herrera, F., Ślęzak, D., Benítez, J.M., et al.: Implementing algorithms of rough set theory and fuzzy rough set theory in the R package "roughsets". Inf. Sci. **287**, 68–89 (2014)
42. Janusz, A., Stawicki, S., Szczuka, M., Ślęzak, D.: Rough set tools for practical data exploration. In: Ciucci, D., Wang, G., Mitra, S., Wu, W.-Z. (eds.) RSKT 2015. LNCS, vol. 9436, pp. 77–86. Springer, Cham (2015). doi:10.1007/978-3-319-25754-9_7
43. Barga, R., Fontama, V., Tok, W.H., Cabrera-Cordon, L.: Predictive Analytics with Microsoft Azure Machine Learning. Springer, Heidelberg (2015)
44. Souza, C.R.: The Accord.NET framework, SãoCarlos, Brazil (2014). http://accord-framework.net. Accessed 11 Feb 2017