MR – RANDOM FOREST ALGORITHM FOR DISTRIBUTED ACTION RULES DISCOVERY

Angelina A. Tzacheva, Arunkumar Bagavathi, Punniya D. Ganesan

Dept. of Computer Science, Univ. of North Carolina at Charlotte, Charlotte, NC, USA
aatzache@uncc.edu, abagavat@uncc.edu, pganesa1@uncc.edu

ABSTRACT

Action rules, which are the modified versions of classification rules, are one of the modern approaches for discovering knowledge in databases. Action rules allow us to discover actionable knowledge from large datasets. Classification rules are tailored to predict the object's class. Whereas action rules extracted from an information system produce knowledge in the form of suggestions of how an object can change from one class to another more desirable class. Over the years, computer storage has become larger and also the internet has become faster. Hence the digital data is widely spread around the world and even it is growing in size such a way that it requires more time and space to collect and analyze them than a single computer can handle. To produce action rules from a distributed massive data requires a distributed action rules processing algorithm which can process the datasets in all systems in one or more clusters simultaneously and combine them efficiently to induce single set of action rules. There has been little research on action rules discovery in the distributed environment, which presents a challenge. In this paper, we propose a new algorithm called MR – Random Forest Algorithm to extract the action rules in a distributed processing environment.

KEYWORDS

Data Mining, Action Rules, MapReduce, Hadoop

1. INTRODUCTION

The process of refining knowledge or finding interesting patterns in large datasets is called Data Mining. Some of the classical knowledge discovery algorithms that can be used to find these patterns are classification, association and clustering. These algorithms identify the enormous number of patterns from the data regardless of the fact that most of the patterns are not of user's interest. Therefore, these patterns are overwhelming to the users to analyze them manually to infer solutions for problems in their corresponding domains. For example, in a cancer dataset, if an oncologist wants to find what changes among the attributes in the data can cure a type of cancer, the classical style of knowledge discovery does not help.

However, there has been research like the algorithm proposed in [1], [2], [3], [4] and [5] in a past decade for an optimistic approach to produce actions (change in an object's state) based on the discovered patterns, which is called an action rule. Action rule is a rule extracted from a dataset that describes the possible transition of objects from one state to another with respect to the distinguished attribute called decision attribute [1]. To generate action rules, the attributes in the dataset are split into two groups called – *Flexible Attributes* and *Stable Attributes*. Flexible attributes are those for which the state can change and the stable attributes are those for which the state is always fixed. To obtain action rules, the data must contain one or more flexible attributes.

Assume the dataset is in the form of an information system $S = \{A \cup B \cup D\}$, where A is a stable attribute and $\{B, D\}$ are flexible attributes out of which D is a distinguished attribute called *decision attribute*. Also assume $\{a_1, a_2, ..., a_n\} \subseteq A, \{b1, b2, ..., bn\} \subseteq B$ *and* $\{d1, d2\} \subseteq D$. The following is an example action rule if the user desires the decision attribute value to change from d_1 to d_2:

$$r_1 = (A, a_2 \rightarrow a_1) \wedge (B, b_1) \Rightarrow (D, d_1 \rightarrow d_2)$$

The above action rule r_1 means that, if the attribute A changes its value from $a1$ to $a2$ and the attribute B remains unchanged value=$b1$, then the attribute D is expected to change from $d1$ to $d2$. $(A, a_2 \rightarrow a_1)$ and (B, b_1) are generally called *atomic action sets*.

If the dataset in small, a single computer system generates action rules faster. However, if the dataset is massive, it becomes very expensive for a single system to produce action rules. Therefore, we propose an algorithm which runs at the same time on several computer systems (nodes). In addition to producing the results faster, the generated action rules are more reliable, as they are cross referenced among the multiple nodes. In this work, in order to generate action rules in a distributed environment, we are using the strategy proposed by Google [12] called MapReduce. For running the MapReduce, we are using Apache's open source Hadoop framework [11] for processing immense datasets in multiple nodes in one or more clusters.

In this work, we propose MapReduce (MR) – Random Forest algorithm for Action Rules discovery. We adapt the algorithm known as Action Rules Discovery Based on Grabbing Strategy (ARoGS) and Learning from Example based on Rough Sets (LERS) [7] to the distributed MapReduce environment. LERS is utilized as a base algorithm to generate action rules in Mappers of the Hadoop cluster. Next, the generated output is fed to the ensemble learning method known as Random Forest [13] in Reducers of the Hadoop cluster to produce a singleton set of action rules. Finally, we employ a second method for generating action rules, called Association-Action Rules (AAR) as described in paper [10], and compare the resulting action rules from Grabbing Strategy and from Association-Action rules Strategy in a distributed environment.

2. Related Work

The notion of action rules was initially proposed in [1] which brings in the idea of splitting the attributes into stable attributes, flexible attributes and a decision attribute. Action rules have been investigated further in [2], [3], [4], [5], [6], [7], [8], [9] and [10]. However, in the earlier research, the action rules were produced using pairs of classification rules. Ras and Dardzinska [5] made the first attempt to give a lattice-theory type of framework for producing the action rules with single classification rule without any detailed algorithm, but directly from the database. Later, Ras and Wyrzykowska [3] gave a LERS [15] type of algorithm that considers only marked certain rules to construct the action rules. Ras and Wyrzykowska [7] proposed a new algorithm named ARoGS which combines each action rule generated from single classification rule with the remaining stable attributes to offer more action rules. This work discovers more action rules that were missed out in the previous algorithms.

Ras and Tsay [6] defined DEAR systems which proposes a definition to calculate support and confidence for the generated action rules. Tzacheva and Ras [9] provided more reliable formula for calculating support and confidence of action rules. The support and confidence of action rules were further simplified and improved in the research of Tzacheva [17], including the introduction of a new measure called Utility. Ras and Dardzinska [10] suggested an Apriori-like algorithm for action rules discovery, which generates association-type action rules using frequent action items. This is one of the methods to generate action rules without using classification rules.

In this work, we combine the ARoGS [7] algorithm with the new support and confidence measures by Tzacheva [17], and we propose a Random-Forest [13] based algorithm for discovery of action rules in a distributed environment utilizing MapReduce, called MapReduce (MR) – Random Forest algorithm for Action Rules discovery. Further, we adapt the Association–Action rules [10] algorithm to our distributed environment. Finally, we compare the results of ARoGS and Association-Action rules operating in the distributed environment.

2. METHODOLOGY

We implement the proposed MR - Random-Forest algorithm for distributed action rules discovery using Apache Hadoop framework [11] and Google MapReduce [12]. An overview of the proposed algorithm is shown on Figure 1. We take as an input a set of files: the data, the attribute names, user specified parameters such as: minimum support, and confidence tresholds, stable attribute names, flexible attribute names, decision attribute choice, decision attribute value to *change_from*, and decision attribute value to *change_to*, which is the desired value of decision attribute (desired object state). We import these input files into the HDFS (Hadoop Distributed File System). Our MapReduce algorithm consists of three jobs: Job 1 runs LERS and ARoGS methods to generate Action Rules (AR); Job 2 runs Association–Action rules method to produce an Association type of Action Rules (AAR); and the Job 3 collects the output of Job1 and Job2 and compares the rules. Job 3 compares both action rules and association action rules, finds the rules which are identical from Job1 and Job2, and produces a single list of rules as an output.

Now we describe the LERS, ARoGS, and Association-Action rules methods in detail. Consider an information system *S*:

$S = (X, A, V_A)$ where,
X is a set of objects: $X = \{x_1, x_2, x_3, x_4, x_5\}$
A is a set of attributes: $A = \{a, b, c, d\}$ and
V_A represents a set of values for each attribute in A. For example $V_b = \{b_0, b_2\}$.

We use the sample information system *S* is shown in Table 1. to demonstrate output from these algorithms. Consider attribute *b* to be a *Stable Attribute* , attributes *{a, c}* to be *Flexible Attributes*, attribute *d* to be the *Decision Attribute*, and that the user desires the decision value to change from d_1 to d_2. Also user is interested in action rules with minimum support of 2 and minimum confidence of 80%.

Table 1. Sample Information System *S*

X	**a**	**b**	**c**	**d**
x_1	a_1	b_0	c_2	d_1
x_2	a_0	b_0	c_1	d_2
x_3	a_2	b_0	c_2	d_1
x_4	a_0	b_0	c_2	d_2
x_5	a_1	b_2	c_2	d_1

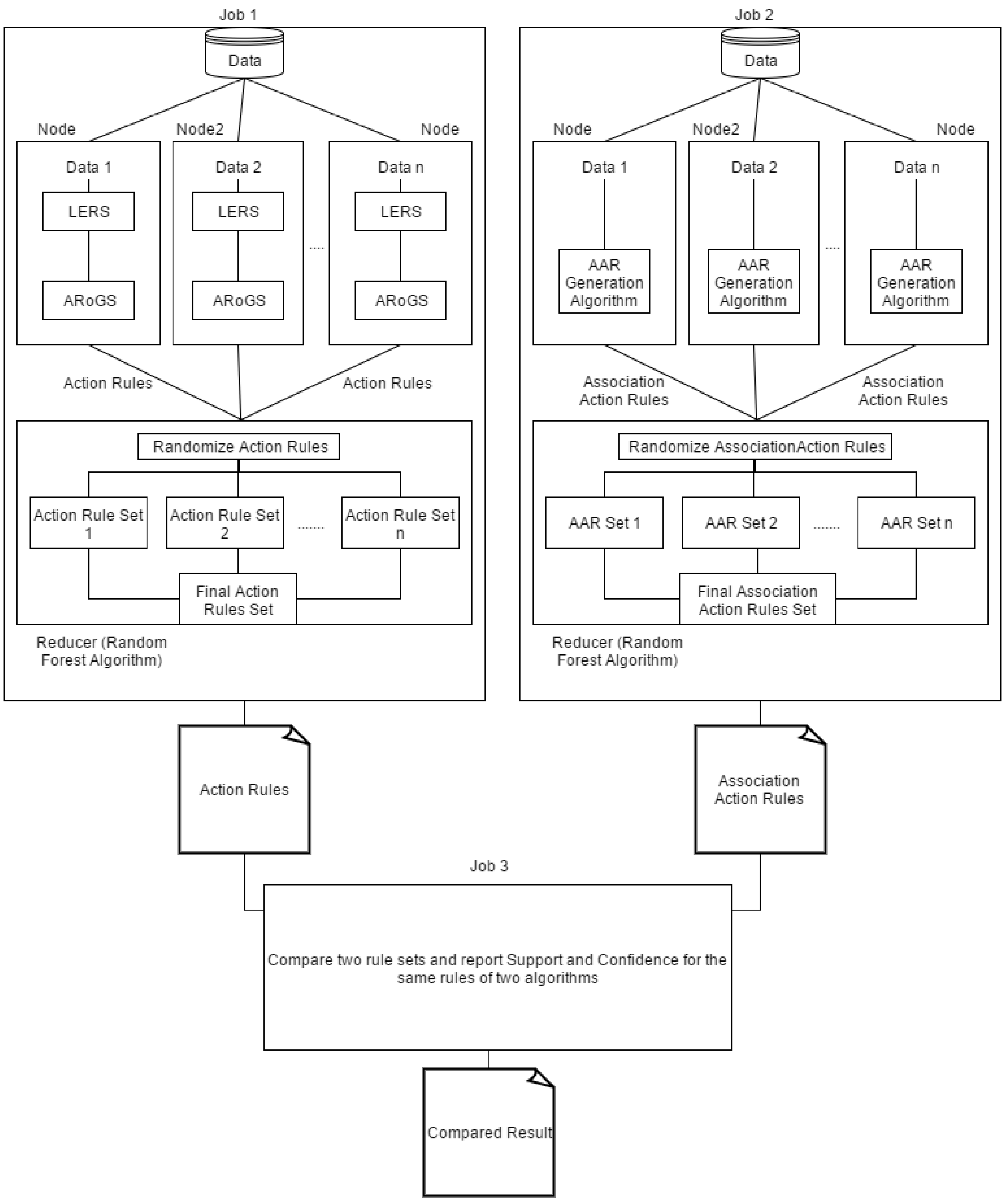

Figure 1. MR-Random Forest Algorithm for Distributed Action Rules Discovery Overview

3.1. LERS

Our proposed implementation of the LERS (Learning from Examples based on Rough Sets) method in a distributed scenario using MapReduce is illustrated in Figure 2. Using the information system S from Table 1., LERS strategy can find all certain and possible rules describing decision attribute *d* in terms of attributes *a*, *b*, and *c*.

ALGORITHM 1:
LERS (*attributesSupport*, *decisionSupport*)

```
(where attributesSupport and decisionSupport are maps with distinct attribute values as
keys and their corresponding value is the objects in the information system supporting
them)

fixedSupport ← attributesSupport
while attributesSupport is not empty do
    for each key, value pair in the attributeSupport do
        if value is a subset of one of the values of decisionSupport then
            Add key and decisionValue to certainRules
            (where certainRules is a map with attribute value as a ‘key’ and decision
            attribute value as a ‘value’)
        else
            Add key and value to possibleRules
            (where possibleRules is a map with attribute value as a ‘key’ and decision
            attribute value as a ‘value’)
        end
        delete key from the attributesSupport
    end
    for each $key_1$, $value_1$ pair in the possibleRules do
        for each $key_2$, $value_2$ pair in the fixedSupport do
            if $key_1$ contains $key_2$ then Continue
            else
                $key_3$ ← ($key_2$, $key_1$)
                $value_3$ ← Set of objects in information system supporting $key_3$
                Add $key_3$ and $value_3$ to attributeSupport
            end
        end
    end
end
```

Figure 2. LERS (Learning from Examples based on Rough Sets) Algorithm in a distributed environment using MapReduce

LERS can be used as a data strategy to generate decision rules. From selected pairs of these decision rules, the action rules can be composed as described by Ras and Wyrzykowska [3], [15]. We consider only marked certain rules to construct the action rules. Since LERS follows bottom-up strategy, it constructs rules with a conditional part of length *x*, then it continues to construct rules with a conditional part of length *x+1*. According to papers [3] and [15], the LERS system rules that get induced from lower and upper approximations are called certain and possible rules, respectively.
Using the information system S from Table 1., the LERS algorithm produces the certain and possible rules at each iteration shown in Table 2. Next, these rules are given as an input to the AR (Action Rules) algorithm, which builds action rules by taking all certain rules from Table 2. The proposed AR algorithm in a distributed environment is illustrated in Figure 3.

decisionSupport: $(d_1)^* = \{x_1, x_3, x_5\}$ and $(d_2)^* = \{x_2, x_4\}$

Table 2. Certain and Possible Rules produced by LERS algorithm on data *S* from Table 1.

Itera tion	**Attribute value support**	**Certain rules**	**Possible rules**
1	$(a_0)^* = \{x_2, x_4\}$ – marked $(a_1)^* = \{x_1, x_5\}$ – marked $(a_2)^* = \{x_3,\}$ – marked $(b_0)^* = \{x_1, x_2, x_3, x_4\}$ $(b_2)^* = \{x_5\}$ - marked $(c_1)^* = \{x_2\}$ - marked $(c_2)^* = \{x_1, x_3, x_4, x_5\}$	$a_0 \rightarrow d_2$ $a_1 \rightarrow d_1$ $a_2 \rightarrow d_1$ $b_2 \rightarrow d_1$ $c_1 \rightarrow d_2$	$b_0 \rightarrow d_1$ $b_0 \rightarrow d_2$ $c_2 \rightarrow d_1$ $c_2 \rightarrow d_2$
2	$(b_0, c_2)^* = \{ x_1, x_3\, x_5\}$		$b_0 \wedge c_2 \rightarrow d_1$ $b_0 \wedge c_2 \rightarrow d_2$

3.2. ARoGS

ARoGS is Action Rules Discovery Based on Grabbing Strategy, which uses LERS. It is given by Ras and Wyrzykowska in paper [7] as an alternative to system DEAR from paper [6]. ARoGS uses LERS to extract action rules, without the need of verifying the validity of the certain relations. It just has to check if these relations are marked by LERS. By using LERS in the pre-processing module for defining classification rules, the overall complexity of ARoGS algorithm decreases.
In our proposed method, we take the final set of certain rules extracted by LERS and create new action rule by combining a certain rule with other certain rules. Using the flexible attributes in the certain rules, atomic action sets like $(a, a_1 \rightarrow a_2)$ can be formed. We extract all action rules, which imply $d_1 \rightarrow d_2$ by using AR algorithm described in Figure 3.

Consider the following action rules, which are obtained by following the algorithm AR using the information system in Table 1:

$$ar_1\ (d_1 \rightarrow d_2) = (a, 1 \rightarrow 0) \Rightarrow (d, 1 \rightarrow 2)$$
$$ar_2\ (d_1 \rightarrow d_2) = (a, 2 \rightarrow 0) \Rightarrow (d, 1 \rightarrow 2)$$

The algorithm ARoGS runs on each action rule generated by algorithm AR, and it produces the following additional action rule (ar_3):

$$ar_3\ (d_1 \rightarrow d_2) = (b, 2) \wedge (a, 1 \rightarrow 0) \Rightarrow (d, 1 \rightarrow 2)$$

ARoGS produces this additional rule, because it is treating each action rule describing the target decision value as a seed and grabs other action rules describing non-target decision values in order to form a cluster. From the newly formed clusters, it builds decision rules, where a grabbed seed is only compared with that seed. Our proposed implementation of ARoGS in a distributed environment is shown on Figure 4.

```
ALGORITHM 2:
AR (certainRules, decisionFrom, decisionTo)
    (where certainRules is a map provided by the algorithm LERS)

        for each $key_1$, $value_1$ pair in the certainRules do
            for each $key_2$, $value_2$ pair in the certainRules do
                if $value_1$ equals decisionFrom and $value_1$ equals decisionTo then
                    if $key_2$ attributes are a subset of $key_1$ attributes and $key_2$ stable
                    attributes are a subset of $key_1$ stable attributes then
                        actions ← empty list
                        for each attribute value $a_1$ in $key_1$ do
                            for each attribute value $a_2$ in $key_2$ do
                                if $a_1$ and $a_2$ belongs to same attribute then
                        a←attributeName($a_1$)
                        actions. Add("($a$, $a_1 \rightarrow a_2$)")
                                end
                            end
                        end
                        Output actions as action rule
                        ARoGS(actions, decisionFrom, decisionTo)
                end
            end
        end
```

Figure 3. AR (ActionRules) Algorithm in a distributed environment using MapReduce

3.3. Association Action Rules

The Association–Action Rules described by Ras and Dardzinska in paper [10] is an algorithm intended to simplify the action rules construction by employing the 'lowest cost' strategy. The Association–Action Rules (AAR) algorithm uses a different approach, from the ones described above, as it generates association-type action rules using *frequent action sets* in an Apriori-like fashion. The extracted action rules are intended to have minimal attribute involvement. The *frequent action set* generation is divided in two steps: merging step and pruning step. In the merging step: we merge the previous two frequent action sets into a new action set. For our example, using the data from the Information System in Table 1, the primary action sets generated by AAR are shown in Table 3. The *frequent action sets* generated by AAR are shown in Table 4. In the pruning step: we discard the newly formed action set if it does not contain the decision action (e.g. the user desired value of decision attribute). In our example, the action set is discarded if $(d, 1 \rightarrow 2)$ is not present in it. From each *frequent action set*, the association action rules are formed. Therefore, the AAR algorithm generates frequent action sets and forms the association action rules from these action sets. Our proposed implementation of AAR algorithm in a distributed environment is shown in Figure 5.

For our example, using the data from the Information system in Table 1, the AAR algorithm generates following Association Action Rules:

$aar_1 (d_1 \rightarrow d_2) = (a, 2 \rightarrow 0) \Rightarrow (d, 1 \rightarrow 2)$
$aar_2 (d_1 \rightarrow d_2) = (a, 1 \rightarrow 0) \Rightarrow (d, 1 \rightarrow 2)$
$aar_3 (d_1 \rightarrow d_2) = (b, 0) \wedge (a, 1 \rightarrow 0) \Rightarrow (d, 1 \rightarrow 2)$
$aar_4 (d_1 \rightarrow d_2) = (c, 0) \wedge (a, 1 \rightarrow 0) \Rightarrow (d, 1 \rightarrow 2)$
……..

……..
aar_{n-1} $(d_1 \rightarrow d_2) = (b, 0) \wedge (c, 0) \wedge (a, 1 \rightarrow 0) \Rightarrow (d, 1 \rightarrow 2)$
aar_n $(d_1 \rightarrow d_2) = (b, 0) \wedge (a, 1 \rightarrow 0) \wedge (c, 2 \rightarrow 1) \Rightarrow (d, 1 \rightarrow 2)$

```
ALGORITHM 3:
ARoGS (actions, decisionFrom, decisionTo)
    (where 'actions' is a list of actions from Algorithm AR)

    stableValues ← list of stable attribute values in actions
    actionsSupport ←set of objects in the information system supporting all attribute values in actions
    missingValues ←set of missing flexible attribute values of the attributes in actions
    and a set of stable attributes values of stable attributes not present in actions
    for each value in missingValues do
        newValues ← combine value with stableValues
        newSupport ← set of objects in the information system supporting newValues in actions
        if newSupport is a subset of actionsSupport then
            Add value to actions
            Output actions as action rule
        end
    end
```

Figure 4. ARoGS (Action Rules Discovery based on Grabbing Strategy) in a distributed environment using MapReduce

Table 3. Primary Action Sets for Information System *S* from Table 1.

Attribute	**Primary action sets**
a	(a, a_0), (a, a_1), (a, a_2), $(a, a_0 \rightarrow a_1)$, $(a, a_0 \rightarrow a_2)$ $(a, a_1 \rightarrow a_0)$, $(a, a_1 \rightarrow a_2)$, $(a, a_2 \rightarrow a_0)$, $(a, a_2 \rightarrow a_1)$
b	(b, b_0) (b, b_2)
c	$(c, c_1 \rightarrow c_2)$ $(c, c_2 \rightarrow c_1)$
d	$(d, d_1 \rightarrow d_2)$ $(d, d_2 \rightarrow d_1)$

Table 4. Frequent Action Sets for Information System *S* from Table 1.

Iteration #	Frequent action sets
1	$(a, a_0) \wedge (d, d_1 \rightarrow d_2)$ $(a, a_1) \wedge (d, d_1 \rightarrow d_2)$ $(a, a_2) \wedge (d, d_1 \rightarrow d_2)$ $(b, b_0) \wedge (d, d_1 \rightarrow d_2)$ $(b, b_2) \wedge (d, d_1 \rightarrow d_2)$ $(a, a_0 \rightarrow a_1) \wedge (d, d_1 \rightarrow d_2)$ $(a, a_0 \rightarrow a_2) \wedge (d, d_1 \rightarrow d_2)$ ……. …….
Iteration #	**Frequent action sets**
2	$(a, a_0) \wedge (b, b_0) \wedge (d, d_1 \rightarrow d_2)$ $(a, a_1) \wedge (b, b_0) \wedge (d, d_1 \rightarrow d_2)$ $(b, b_0) \wedge (c, c_1) \wedge (d, d_1 \rightarrow d_2)$ $(a, a_0 \rightarrow a_1) \wedge (b, b_0) \wedge (d, d_1 \rightarrow d_2)$ ……. …….
Iteration #	**Frequent action sets**
3	$(a, a_0) \wedge (b, b_0) \wedge (c, c_1) \wedge (d, d_1 \rightarrow d_2)$ $(a, a_1) \wedge (b, b_0) \wedge (c, c_1) \wedge (d, d_1 \rightarrow d_2)$ $(a, a_2) \wedge (b, b_0) \wedge (c, c_1) \wedge (d, d_1 \rightarrow d_2)$ $(a, a_0) \wedge (b, b_0) \wedge (c, c_1 \rightarrow c_2) \wedge (d, d_1 \rightarrow d_2)$ ……. …….

3.4. Support and Confidence of Action Rules

Consider an action rule *R* of the form

$(Y_1 \rightarrow Y_2) \Rightarrow (Z_1 \rightarrow Z_2)$ where,

Y is concatenation of all action sets that support the decision action *Z*

Y_1 = attribute values on left side of all actions in the left side of the action rule *R*

Y_2 = attribute values on right side of all actions in the left side of the action rule *R*

Z_1 = decision attribute value on the left side

Z_2 = decision attribute value on the right side

1) *Support and Confidence: Association Action Rules*

For an Association Action Rule *aar*, the following support and confidence applies, given in paper [9]:

$Support\ (aar) = \min\ [\text{card}\ (Y_1 \wedge Z_1), \text{card}\ (Y_2 \wedge Z_2)]$
$Confidence\ (aar) = [card\ (Y_1 \wedge Z_1) / card\ (Y_1)] * [card\ (Y_2 \wedge Z_2) / card\ (Y_2)]$

where $card(Y_1) \neq 0$ and $card(Y_2) \neq 0$

2) *Support and Confidence: ARoGS*

In ARoGS support and confidence of an action rule *ar* are calculated using the following formulas given in paper [7]:

$$Support\ (ar) = card\ (Y_2 \wedge Z_2)$$
$$Old\ Confidence\ (ar) = [card\ (Y_1 \wedge Z_1) / card\ (Y_1)] * [card\ (Y_2 \wedge Z_2) / card\ (Y_2)]$$

In our proposed method, this confidence is replaced by the following confidence formula given by Tzacheva et al. [17] to reduce complexity:

$$New\ Confidence\ (ar) = [card\ (Y_2 \wedge Z_2) / card\ (Y_2)]$$

where $card(Y_1) \neq 0$ and $card(Y_2) \neq 0$

In the above formulas, *card (X)* means Cardinality which is the number of objects in the information system containing the value X. The algorithms eliminate action rules if the corresponding support and confidence is less than the given minimum support and confidence. For example, for the rule $ar_3\ (d_1 \rightarrow d_2)$ and $aar_3\ (d_1 \rightarrow d_2)$, the Support = 0 which is less than the user specified support threshold = 2 in our example for the Information System *S* in Table 1. Therefore, these rules are discarded by the algorithms.

```
ALGORITHM 5: Reduce (Key, values)
    (where Key is an action rule from Algorithm AR or Algorithm AAR and values is a
    list of support and confidences of a Key from n Maps)
    if Count(values) >= n / 2 then
        supp ← Average of all supports
        conf ← Average of all confidences
        if supp >= minimum support and conf >= minimum confidence then
            Output Key with supp and conf
        end
    end
```

Figure 6. Random Forest algorithm in Reduce part of MapReduce combines Action Rules from multiple Mappers

3.5. MR-Random Forest Algorithm for Action Rules

In our proposed implementation using the Hadoop MapReduce framework, the above described algorithms run in parallel in distinct threads as two separate jobs, as shown on Figure 1. LERS and AR in Job1, and AAR in Job2. Each job has its own Map and Reduce parts. The LERS, AR, and AAR algorithms are implemented in the Map part. Hadoop splits the data and gives splits of data to several Map parts (Mappers). The resulting action rules from all the Mappers are combined in such a way that the *action rule* acts as a *key* and the *support and confidence* from all the Mappers acts as *iterator* list of values. The combined action rules are given to the Reduce part, where we propose using a Random Forest [13] type of algorithm in order to combine the output from all the Mappers. The Random Forest algorithm works in analogy to ‘voting’, where if more than 50% of the parties agree, the vote is accepted. In our proposed implementation, the

Random Forest algorithm checks the output from all the Mappers, and if it finds an action rule which is generated from more than 50% of the Mappers it retains that action rule. If so, it averages all supports and confidences from these Mappers for the given action rule. Then, it checks the averaged support and confidence against the minimum support and confidence thresholds specified by the user. If the support and confidence thresholds are met, the action rule is retained, and included in the final list of action rules, produced as an output from this system, and presented to the user. Our proposed MR-Random Forest Algorithm, implemented in the *Reduce* part of MapReduce, is shown on Figure 6. This figure gives an overview of how our Reduce part works.

3. EXPERIMENT AND RESULTS

We used two datasets for testing our proposed MR - Random-Forest algorithm for distributed action rules discovery: Car Evaluation dataset and Mammographic-mass dataset, obtained from the Machine Learning Repository by Information and Computer Sciences of the University of California, Irvine [16].

We ran the ARoGS and AAR (Association Action Rules) algorithms on the University of North Carolina at Charlotte Hadoop Research cluster, which has 73 nodes. Hadoop splits the data with respect to its block size. Even though the default block size in Hadoop is 64 MB, it can be reduced to support smaller datasets. The minimum block size we can set is 1.04 MB. Since the minimum block size in Hadoop is 1.04 MB, it would not be splitting our original data. As we are adapting the Action Rules discovery algorithm to work witch much bigger datasets, than it has worked with before, then we replicate the original datasets multiple times to test the proposed algorithm in a distributed environment. This also brings the final dataset to size greater than 1.04 MB, so Hadoop splits it automatically.

We chose the Car Evaluation dataset, and the Mammographic-mass dataset for this study, in order to illustrate the application of Action Rules in two different domains: transportation domain, and medical domain.

Table 5. Properties of Car Evaluation dataset and Mammographic-Mass Dataset

Property	**Car Evaluation Dataset**	**Mammographic Mass Dataset**
Number of instances	1728	961
Replication Factor	116	518
Number of instances after replication	200448	497798
Attributes	7 attributes • Buying • Maintenance • Doors • Persons • Luggage boot • Safety • Class	6 attributes •BI-RADS assessment • Patient's age • Shape • Margin • Density • Severity
Decision attribute values	Class (unacc, acc, good, vgood)	Severity (0 - benign, 1 - malignant)
Original data size	52.3 Kilo Bytes	16 Kilo Bytes
Final data size	5.92 Mega Bytes	7.83 Mega Bytes

The Car Evaluation dataset [16] is donated by Prof. Dr. Marko Bohanec, from Department of Knowledge Technolgoies, Jozef Stefan Institute, in Liublijana, Slovenia. It is intended to evaluate cars according to the car acceptability, according to its buying price, maintenance cost, technical characteristics such as comfort, number of doors, number of persons to carry, the size of its luggage boot, and the car safety. The Car Evaluation dataset has 1728 tuples, and 7 attributes, as shown in Table 5. For the purpose of this study, the Car Evaluation dataset was replicated 116 times, in order to increase its size, and demonstrate the scalability of our proposed method. Action Rules extracted for this dataset can suggest actions to be undertaken (changes in flexible attributes) if the user would like to increase the car's safety, or if the user would like to change the car state from 'unacceptable' (unacc) to 'acceptable' (acc). An example Action Rule extracted from this dataset is:

> *ar_{Car1} (class, unacc → acc) = (buying, buyinglow→buyinglow) ^ (persons, persons2 → persons4) ^ (safety, → safetyhigh) ➔ (class, unacc → acc) [Support: 237 & Confidence: 93.0%]*

The rule *ar_{Car1}* means that: if the buying price of the car remains low (*buyinglow*), and the number of persons it can carry increases from 2 (*persons2*) to 4 (*persons4*), and the safety of the car increases from any value to high (*safetyhigh*), then the decision attribute (*class*) value is expected to change from unacceptable (*unacc*) to acceptable (*acc*). A total of 237 tuples (objects) support this rule, and we are 93% confident in the validity of this rule. Example Actions, called Meta-Actions, which can trigger the above changes are: '*improve air bags*' (to increase safety); '*improve breaks*' (to increase safety); '*make larger salon*' (to increase person capacity of the vehicles). These are called Meta-Actions as described by Tzacheva and Ras [9], since they trigger the suggested changes in flexible attributes specified by the Action Rules. The Meta-Actions can either be provided by expert in the domain and added to the original data to augment it, or they can be automatically extracted from text descriptions associated with the data as shown by Kuang and Ras [18]. For this study, the attributes *{Buying, Maintenance, Doors}* are designated as *Stable Attributes*, and the attributes *{Persons, LuggageBoot, Saftety}* are designated as *Flexible Attributes*, and the attribute *Class* is designated as the *decision attribute*, which is also a flexible attribute. These parameters are shown in Table 6.

The Mammographic-Mass dataset [16] is donated by Prof. Dr. Rdiger Schulz-Wendtland from the Institute of Radiology at the University Erlangen-Nuremberg, Germany. This dataset is used to predict the severity (benign or malignant) of a mammographic mass lesion from BI-RADS attributes and the patient's age. It contains a BI-RADS assessment, the patient's age and three BI-RADS attributes together with the ground truth (the severity field) for 516 benign and 445 malignant masses that have been identified on full field digital mammograms collected at the University Erlangen-Nuremberg. The Mammographic-Mass dataset contains 961 instances, and has 6 attributes, as shown in Table 5. For the purpose of this study, the Car Evaluation dataset was replicated 518 times, in order to increase its size, and demonstrate the scalability of our proposed method. Action rules extracted from the Mammographic-Mass dataset can suggest actions to be undertaken (changes in flexible attributes), in order to re-classify a mammographic mass lesion (tumor) from class: malignant to class: benign. An example Action Rule extracted from this dataset is:

> *ar_{Mam1} (severity, 1 → 0) = (Margin, 3→4) ^ (BI-RADS, 5 → 4) ^ (Density, → 3) ➔ (severity, 1 → 0) [Support: 284 & Confidence: 82.4%]*

The rule *ar_{Mam1}* means that: if the *Margin* of the lesion (tumor) changes from 3 to 4, and the *BI-RADS* assessment changes from 5 to 4, and the *Density* of the lesion (tumor) changes from any value to 3, then the *severity* (decision attribute) is expected to change from value 1 (*malignant*) to value 0 (*benign*). A total of 284 tuples (objects) support this rule, and we are 82.4% confident in the validity of this rule. The suggested desired changes can be triggered by Meta-Actions [9]. Example Meta-Actions, which can trigger the above changes are: '*doctor prescribes specific medication*' (to change BI-RADS assessment); or '*doctor performs a specific medial procedure*' (to change the margin of the lesion). For this study, we designate {*BIRADS*, *Margin*, *Density*, *Shape*} as *Flexible Attributes*. We designate {*Shape*, *Age}* as a *Stable Attributes*. We designate *Severity* as our *decision* (class) attribute, which is also a flexible attribute. These parameters are shown in Table 6.

Table 6. Parameters used for Action Rules discovery on the Car Evaluation dataset and Mammographic-Mass dataset

Parameters	**Car Evaluation Dataset**	**Mammographic-Mass Dataset**
Stable attributes	Buying, Maintenance, Doors	Shape, Age
Expected decision action	(Class) unacc → acc	(Severity) 1 → 0
Minimum support and confidence	150, 80%	50, 70%

Since we replicated the datasets multiple times, as shown in Table 5., the size of the data was substantially increased from the original. Next, we ran our experiment, and Hadoop made 6 splits of the data for the Car Evaluation dataset, and it made 8 splits of the data for the Mammographic-mass dataset. The ARoGS algorithm took 1.84 minutes to process the Car Evaluation data on a single node, and it took 1.12 minutes to process the Car Evaluation dataset on 6 nodes. The Association Action Rules algorithm took 11.09 minutes to process the Car Evaluation dataset on a single node, and it took 5.4 minutes to process the Car Evaluation dataset on 6 nodes. The ARoGS algorithm took 0.53 minutes to process the Mammographic Mass dataset on a single node, and it took 0.29 minutes to process the Mammographic Mass dataset on 8 nodes. The AAR algorithm took 9.4 minutes to process the Mammographic Mass dataset on a single node, and it took 5.4 minutes to process the Mammographic Mass dataset on 8 nodes. A comparison of the processing time for these algorithms is shown on Table 7.

Table 7. Comparison of processing time for ARoGS and AAR algorithms using MapReduce (MR) – Random Forest method on Hadoop

Dataset	**Number of splits (nodes)**	**ARoGS (minutes)**	**AAR (minutes)**
Car Evaluation Data	1	1.84	11.09
	6	1.12	5.4
Mammographic-Mass Data	1	0.53	9.4
	8	0.29	6.2

The processing times shown in Table 7. indicate that: the larger the data size is, the faster our algorithms run (both ARoGS and AAR algorithms), when using multiple nodes (in a distributed environment with MapReduce framework), compared to a single node (a single machine). From the results in Table 7., we can also see that ARoGS algorithm generates the Action Rules much faster than the AAR algorithm does, while using the MR - Random Forest method in the Reduce phase for both. The AAR (Association Action Rules) takes a much longer time to generate Action Rules because it follows Apriori-like method described in section 3.3 to produce all possible combination of *action sets* and from these *action sets*, it generates all possible Action Rules. Table 8. depicts sample comparison of rules generated by both the algorithms on the Car dataset.

Next, we compare the ARoGS and the AAR algorithm. Our results indicate that the ARoGS algorithm produces more general Action Rules, while the AAR algorithm produces more specific Action Rules. By general Action Rule we mean that the rule contains an *atomic action set* like *(safety, -> safetyhigh)* i.e. the *safety* is changed from *any* value to value *safetyhigh*. On the other hand, the AAR algorithm produces only specific Action Rules i.e. the *action sets* have both values *chage_from* and *change_to* specified, such as: *(safety, safetlylow -> safetyhigh)*. Even though the AAR algorithm follows Apriori-like method and takes much longer time to process, it generates more rules comparing to the ARoGS method. For our study, the ARoGS produced 20 Action Rules the Car Evaluation Dataset, while AAR produced 124 Action Rules, out of which 80 rules can be generalized to the rules produced by ARoGS algorithm. We show an example of ARoGS general Action Rule, and its corresponding AAR specific Action Rules on Table 8.
This comparison of Action Rules produced by ARoGS and AAR is performed in Job3 of our proposed method as shown on Figure 1. Job3 produces the final list of Action Rules presented to the user.

Table 8. Comparison of general and specific Action Rules produced by ARoGS and AAR respectively

ARoGS general action rule	**AAR** corresponding specific ation rule
(safety, → safetyhigh) (buying, buyinglow→buyinglow) ^ (maint, maintvhigh → maintvhigh) ^ (persons, persons2 → persons4) ^ *(safety, → safetyhigh)* ➔ (Class, unacc → acc) [Support: 232 & Confidence: 100.0%]	(buying, buyinglow → buyinglow) ^ (maint, maintvhigh → maintvhigh) ^ (persons, persons2 → persons4) ^ *(safety, safetylow → safetyhigh)* ➔ (Class, unacc → acc) [Support: 232 & Confidence: 100%]]
	(buying, buyinglow → buyinglow) ^ (maint, maintvhigh → maintvhigh) ^ (persons, persons2 → persons4) ^ *(safety, safetymed → safetyhigh)* ➔ (Class, unacc → acc) [Support: 232 & Confidence: 100%]]
	(buying, buyinglow → buyinglow) ^ (maint, maintvhigh → maintvhigh) ^ (persons, persons2 → persons4) ^ *(safety, safetyhigh → safetyhigh)* ➔ (Class, unacc → acc) [Support: 232 & Confidence: 100%]]

3. CONCLUSION

In this work, we propose a novel method MR – Random Forest Algorithm for Distributed Action Rules Discovery, which adapts two Action Rules discovery algorithms, ARoGS and AAR, to a distributed environment through Random-Forest approach, using MapReduce framework on Hadoop. The proposed new method presents a highly scalable solution for Action Rules discovery as it adjust to large datasets, through splitting the data, and utilizing multiple nodes for processing. Our results show significant improvement in processing time for Action Rules extraction, with increased data size, when using multiple nodes, compared to the standard single node (single machine) processing. The large datasets are very difficult to process on a single machine using the currently existing Action Rules discovery methods.

Action rules can be used in medical, financial, education, transportation, and industrial domain. Action rules suggest actions (changes in flexible attributes) the user can undertake to accomplish their goal. In our study, example goals were: in transportation domain: '*change the car state from unacceptable to acceptable*'; in medical domain: '*re-classify a breast tumor form malignant to benign severity*'. In other domains example goals can be: in financial domain: '*increase the customer loyalty*'; '*how to decrease the risk of a loan*'; education domain: '*how to improve student evaluations*'. Considering the fact that nowadays all these organizations collect and store large amounts of data, and the fact that the amount of data grows at high rate on daily basis, this study makes an important contribution by adapting the Action Rules discovery algorithms to a distributed environment, using MapReduce and Random Forest approach, therefore making the algorithm highly scalable to handle large amounts of data. Very limited work has been done on adapting Action Rules discovery to a distributed environment processing, therefore this study contributes to solving an important challenge.

As future work, we plan experiments of the proposed MR-Random Forest algorithm for distributed Action Rules discovery with financial, and education datasets, as well as social network data. Future work also includes experiments with Spark distributed environment, as an alternative to MapReduce, because of its capabilities to hold large amounts of data in memory between jobs, which may improve the processing time. In the future we also plan to optimize the AAR (Association Action Rules) algorithm to extract general Action Rules similar to ARoGS, in order to reduce the complexity of AAR algorithm.

ACKNOWLEDGEMENTS

The authors would like to thank Prof. Dr. Mirsad Hadzikadic, director of the Complex Systems Institute at University of North Carolina at Charlotte, for his support.

REFERENCES

[1] Z.W. Ras, A. Wieczorkowska (2000), Action-Rules: How to increase profit of a company, in Principles of Data Mining and Knowledge Discovery, Proceedings of PKDD 2000,Lyon, France, LNAI, No. 1910, Springer, pp. 587-592.

[2] Z. He, X. Xu, S. Deng, R. Ma (2005), Mining action rules from scratch, Expert Systems with Applications, Vol. 29, No. 3, pp. 691-699.

[3] Z.W. Ras, E. Wyrzykowska, H. Wasyluk (2007), ARAS: Action rules discovery based on Agglomerative Strategy, in Mining Complex Data, Post-Proceedings of 2007 ECML/PKDD Third International Workshop (MCD 2007), LNAI, Vol. 4944, Springer, 2008, pp. 196-208.

[4] S. Im, Z.W. Ras. (2008), Action rule extraction from a decision table: ARED. Foundations of Intelligent Systems, Proceedings of ISMIS'08, A. An et al. (Eds.), Springer, LNCS, Vol. 4994, 2008, pp. 160-168.

[5] Z.W. Ras, A. Dardzinska (2006), Action rules discovery, a new simplified strategy, Foundations of Intelligent Systems, LNAI, No. 4203, Springer, pp. 445-453.

[6] Z.W. Ras, L.S. Tsay (2003), Discovering extended action-rules, System DEAR, in Intelligent Information Systems 2003, Advances in Soft Computing, Proceedings of the IIS'2003 Symposium, Zakopane, Poland, Springer, pp. 293-300.

[7] Z. W. Ras, E. Wyrzykowska (2007), ARoGS: Action Rules discovery based on Grabbing Strategy and LERS, in Proceedings of 2007 ECML/PKDD Third International Workshop on Mining Complex Data (MCD 2007), Univ. of Warsaw, Poland, 2007, pp. 95-105.

[8] A.A. Tzacheva, Z.W. Ras (2007), Constraint based action rule discovery with single classification rules, in Proceedings of the Joint Rough Sets Symposium (JRS07), LNAI, Vol. 4482, Springer, pp. 322-329.

[9] A.A. Tzacheva, Z.W. Ras (2010), Association Action Rules and Action Paths Triggered by Meta-Actions, in Proceedings of 2010 IEEE Conference on Granular Computing, Silicon Valley, CA, IEEE Computer Society, pp. 772-776.

[10] Z.W. Raś, A. Dardzińska, L.-S. Tsay, H. Wasyluk (2008), Association Action Rules, IEEE/ICDM Workshop on Mining Complex Data (MCD 2008), Pisa, Italy, ICDM Workshops Proceedings, IEEE Computer Society, 2008, pp. 283-290.

[11] A. Bialecki, M.Cafarella, D.Cutting and O. Omalley (2005), Hadoop: A Framework for running applications on large clusters built of commodity hardware. [http://lucene.apache.org/hadoop] . Vol .11, 2005.

[12] J.Dean and S. Ghemawat (2004), MapReduce: Simplified Dataprocessing on large clusters in proceedings of the 6th conference on Symposium on Operating Systems Design and Implementation – Volume 6, ser. OSDI'04, Berkeley, CA, USA, USENIX Association, 2004, pp.10-10.

[13] L. Breman (2001), Random Forests, in Machine Learning, Vol. 45, Kluwer Academic, 2001, pp. 5-32

[14] X. Lin, (2014), MR-Apriori: Association Rules algorithm based on MapReduce, in 5th IEEE International Conference on Software Engineering and Service Science (ICSESS) 2014, Beijing, China, pp. 141-144.

[15] J. W. Grzymała-Busse, S. R. Marepally, Y. Yao (2013), An Empirical Comparison of Rule Sets Induced by LERS and Probabilistic Rough Classification , in Rough Sets and Intelligent Systems, Vol. 1, Springer, pp. 261-276.

[16] M. Lichman (2013), UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[17] A.A. Tzacheva, C.C. Sankar, S. Ramachandran, R.A. Shankar (2016), Support Confidence and Utility of Action Rules Triggered by Meta-Actions, in proceedings of 2016 IEEE International Conference on Knowledge Engineering and Applications (ICKEA 2016), Singapore.

[18] J. Kuang, Z.W. Ras (2015), In Search for Best Meta-Actions to Boost Business Revenue, in Proceedings of the Conference on Flexible Query Answering Systems 2015, in Krakow, Poland, Advances in Intelligent Systems and Computing, Vol. 400, Springer, 2015, pp. 431-443.

Authors

Angelina A. Tzacheva is a Teaching Associate Professor at the Department of Computer Science at the University of North Carolina at Charlotte. Her research interests include: data mining and knowledge discovery in databases, multimedia and distributed databases, and big data analytics.

Arunkumar Bagavathi is a Ph.D of Computer Science student at the University of North Carolina at Charlotte. He received his M.S. in Computer Science in 2016. His research interests include data mining and knowledge discovery, big data analytics, cloud computing, mobile applications, and social network mining.

Punniya D. Genesan is a M.S. of Data Science and Business Analytics student at the University of North Carolina at Charlotte. He has worked on Data Warehousing, and Business Intelligence projects, Big Data Analytics, Data modelling and Machine Learning.