**MR – Random Forest Algorithm for Distributed Action Rules Discovery**

Department of Computer Science,University of North Carolina at Charlotte

Charlotte, NC, USA

e-mail: amekala@uncc.edu,rloka@uncc.edu,vmohan3@uncc.edu,amarathe3@uncc.edu,skondap3@uncc.edu,cmarriva@uncc.edu

***Abstract*— *In the distributed database system, knowledge in the form of data is stored at different locations. To classify this data into service classes and eventually to report them based on work qualifiers, Classification Rules are used. With the rapid advancements in the field of Data Mining, action rules have been prevalently used instead of classification ruled to discover knowledge from huge databases. As the data generation process is growing fast in each sect of computer science, modelling and analyzing this data to predict the future result of business product is nevertheless an Herculaneum task for Data Scientists. With growing peed of internet and easy distributed data storage strategies, massive amount of data is generated at every moment. To discover knowledge from this distributed data, distributed action rules processing algorithm are needed. In our project, we have used the MR-Random Forest algorithm to extract meaningful knowledge and action rules in distributed environment.***

***Keywords-Data Mining; Action Rules; MapReduce; Hadoop***

1. **INTRODUCTION**

Knowledge without implementation is mere burden. Data Mining field of computer science is flourishing at tremendous speed. In 21st century, 'Data' has become the most precious commodity. It has its applications ranging from health care, Business, Education, Politics etc. Data mining is used to find useful patterns from the available datasets. Traditional ways to discover knowledge from dataset such as Classification, Association and Clustering algorithms doesn't work with huge distributed data environments.

An action rule performs the specific action to be taken when the certain conditions are fulfilled. A simple example of action rule could be understood by ' For, Do While and If-Then' loop. In above cases we define certain conditions, and when those conditions are fulfilled, specific tasks are performed repeatedly. Decision attribute decides the transition of an object from one state to other. Each dataset contains two types of attributes; Stable and Flexible attributes, which further leads to generate action rules. Object can change its state based on flexible attribute while it can't change its state if attribute is stable. Action Rules can't be generated if there is no flexible attribute in the dataset. In other way, flexible attribute is a 'must' condition to generate action rules.

*Let's just for example we assume the dataset as, S = {P ∪ Q ∪ D}, where P is a stable attribute and {Q, D} are flexible attributes out of which D is a distinguished attribute called Decision Attribute. Also assume {p1, p2, …., pn} ⊆ P, {q1, q2, …., qn} ⊆ Q and {d1, d2} ⊆ D. If the decision attribute value is changed from d1 to d2, then the action rule would be as following;*

r1 = (P, p2 → p1) ^ (Q, q1) → (D, d1 → d2)

Single computer system works better on smaller dataset to produce action rules while to generate action rules from a massive amount of dataset, it becomes unfeasible for single computer system. To solve this problem, we have used Map Reduce algorithm in our project. It works well even on distributed environments. We perform the practical part, we have used Hadoop framework to process massive datasets.

In our project, we have used MR- Random Forest Algorithm to discover action rules from distributed datasets. Along with that, we have implemented ARoGS (Action Rules Discovery Based on Grabbing Strategy) and LERS (Learning from Example based on Rough Sets) algorithms for distributed database environment. Further, to generate the singleton set of action rules we give the output of previous method (ARoGS and LERS) to Random forest in Hadoop cluster. Eventually, we also implemented the Association-Action Rules method to generate the action rules and thereafter we compared the results of ARoGS and LERS in distributed dataset environment.

1. **RELATED WORK**

The attributes are grouped under stable, flexible or decision based on action rules proposed initially, all these classifications were made as pairs. Ras and Wyrzykowska came out with a new idea of classifying the action rules with a single rule directly from the database. They both produced an algorithm similar to LERS and then they evolved it and found a new algorithm called ARoGS where they match each action rule with other stable attributes and come up with more action rules.

DEAR systems which helps us to calculate support and confidence was found by Ras and Tsay and later Ras along with Tzacheva found a more reliable way to calculate with a different way called UTILITY. A method to calculate action rules without classification wasfound by Ras and Dardzinska.

Finally, here they produce a new Random forest-based algorithm using ARoGS method and using the new support and confidence proposed by Tzacheva in a distributed environment and comparing the previous result with Association-Action rules, which was named as MapReduce (MR) – Random Forest algorithm.

1. **METHODOLOGY**

We implement the proposed project to find action rules using Hadoop framework.

Hadoop Framework consists of two layers HDFS and MapReduce. HDFS is primarily used for storage of the data, when HDFS takes data, it breaks the information down into separate pieces and distributes them to different nodes in a cluster allowing for parallel processing. Mapreduce is a programming model used to process the data on a cluster of commodity hardware.

Input files for the Action Rules and Association Action Rules are datafile, attribute file and other user defined parameters like minimum threshold limit for support and confidence, stable attribute names, flexible attribute names, choice of decision attribute, decision attribute value to start from, and decision attribute value to end to, which is the desired value of decision attribute (desired object state).

We place the data file and attribute file in Hadoop Distributed File System (HDFS). MapReduce programming model consists of 3 jobs. Job 1 uses LERS (Learning examples from rough sets) and ARogs (Association Rules of grabbed system) to create action rules. Job 2 uses Association Action Rules(AAR) to create action rules. Action Rules generated from job 1 and job 2 are given as input to the job 3, where action rules are compared, and singleton identical list of rules are generated as output. Let us explore LERS, ARoGS, and AssociationAction rules methods in depth. Assume an information system S:

S = (X, A, VA) where,

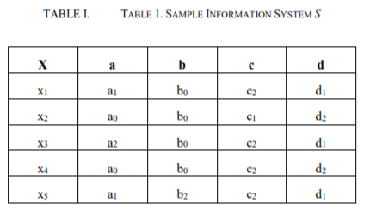
X is a set of objects: X = {x1, x2, x3, x4, x5}

A is a set of attributes: A = {a, b, c, d} and

VA contains the set of values for each attribute present in A.

Consider for example Vb = {b0, b2}.

To illustrate output from these algorithms, we shall use the sample information system S given in Table 1. Let us consider attribute {b} to be a Stable Attribute, attributes {a, c} to be Flexible Attributes and attribute d to be the Decision Attribute. Here, user wants to change the decision attribute value to change from d1 to d2. User can consider to generate action rules, which satisfies the minimum support of 3 and minimum confidence of 80%.



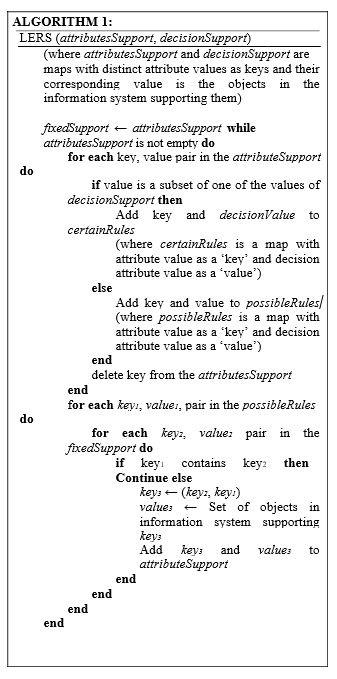


Figure 2. LERS (Learning from Examples based on Rough Sets) Algorithm in a distributed environment using MapReduce

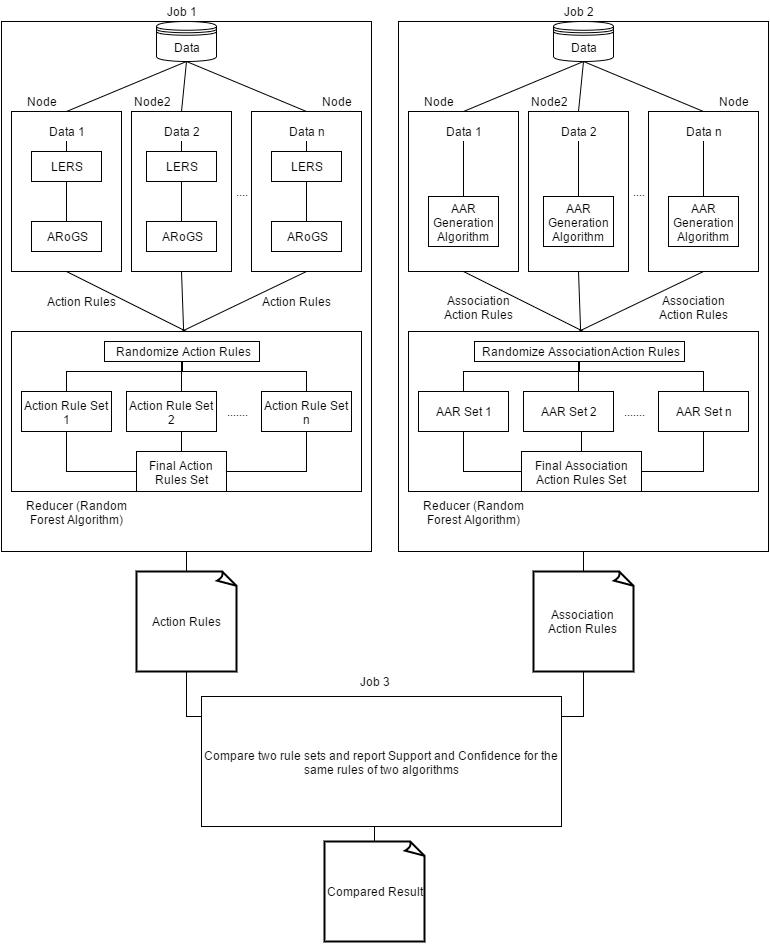


Figure 1. MR-Random Forest Algorithm for Distributed Action Rules Discovery Overview

*A. 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.

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*: (d1) \*= {x1, x3, x5} and (d2) \*= {x2, x4}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Itera** | **Attribute value support** | **Certain** | | **Possible** |
| **tion** |  | **rules** | | **rules** |
|  |  |  |  |  |
| 1 | (a0)\* = {x2, x4} – marked | a0 | → d2 | b0 → d1 |
|  | (a1)\* = {x1, x5} – marked | a1 | → d1 | b0 → d2 |
|  | (a2)\* = {x3,} – marked | a2 | → d1 | c2 → d1 |
|  | (b0)\* = {x1, x2, x3, x4} | b2 → d1 | | c2 → d2 |
|  | (b2)\* = {x5} - marked | c1 | → d2 |  |
|  | (c1)\* = {x2} - marked |  |  |  |
|  | (c2)\* = {x1, x3, x4, x5} |  |  |  |
| 2 | (b0, c2)\* = { x1, x3 x5} |  |  | b0 ^ c2 → d1 |
|  |  |  |  | b0 ^ c2 → d2 |
|  |  |  |  |  |

TABLE II. CERTAIN AND POSSIBLE RULES PRODUCED BY

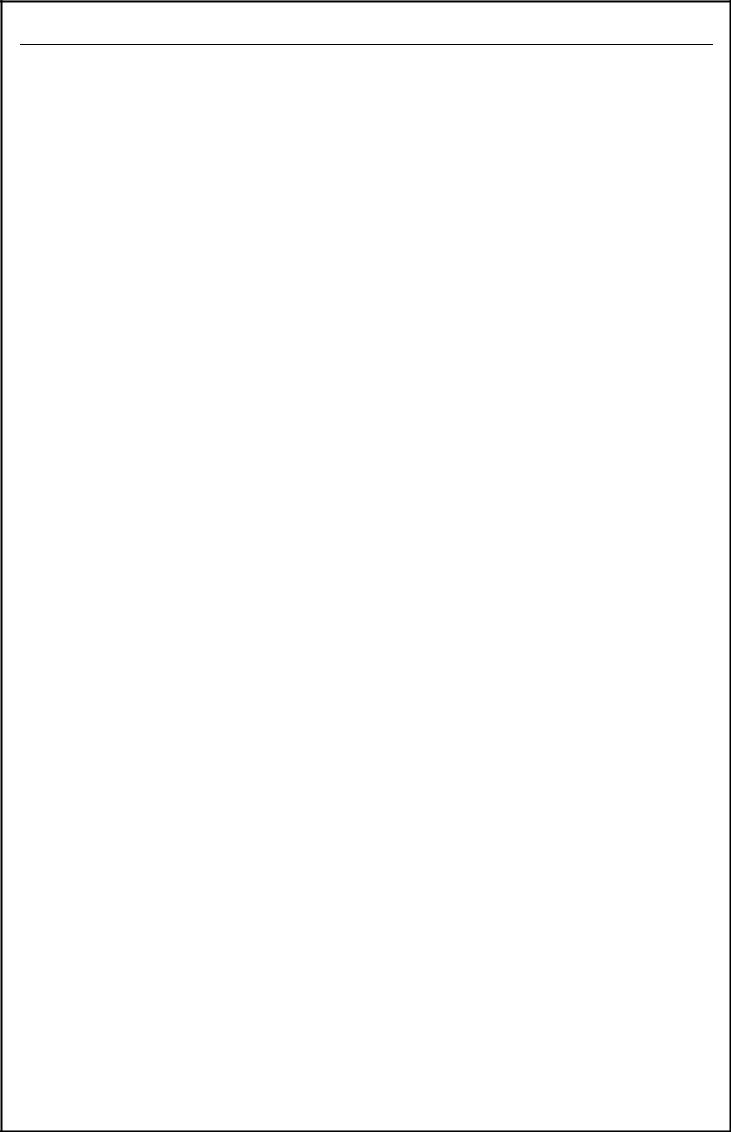
LERS ALGORITHM ON DATA S FROM TABLE 1.

*B. 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, a1 → a2)* can be formed. We extract all action rules,which imply *d1* *→ d2* 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:



**ALGORITHM 2:**

AR (*certainRules*, *decisionFrom*, *decisionTo*)



(where *certainRules* is a map provided by the algorithm *LERS*)

**for each** *key1*,*value1*pair in the*certainRules* **do for each** *key2*,*value2*pair in the

*certainRules* **do**

**if** *value1* **equals** *decisionFrom* **and** *value1* **equals** *decisionTo* **then**

**if** *key2*attributes are a subset of*key1* attributes**and** *key2* stableattributes are a subset of *key1* stable attributes **then**

*actions* ← empty list

**for each** attribute value*a1*in

*key1* **do**

**for each** attribute value

*a2* in *key2* **do**

**if** *a1*and*a2*belongs

to same attribute **then**

*a*←attributeName(*a1*)

*actions*. Add(“(*a*, *a1* → *a2*)”)

**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.

*ar1 (d1 → d2) = (a, 1 → 0) (d, 1 → 2)*

*ar2 (d1 → d2) = (a, 2 → 0) (d, 1 → 2)*

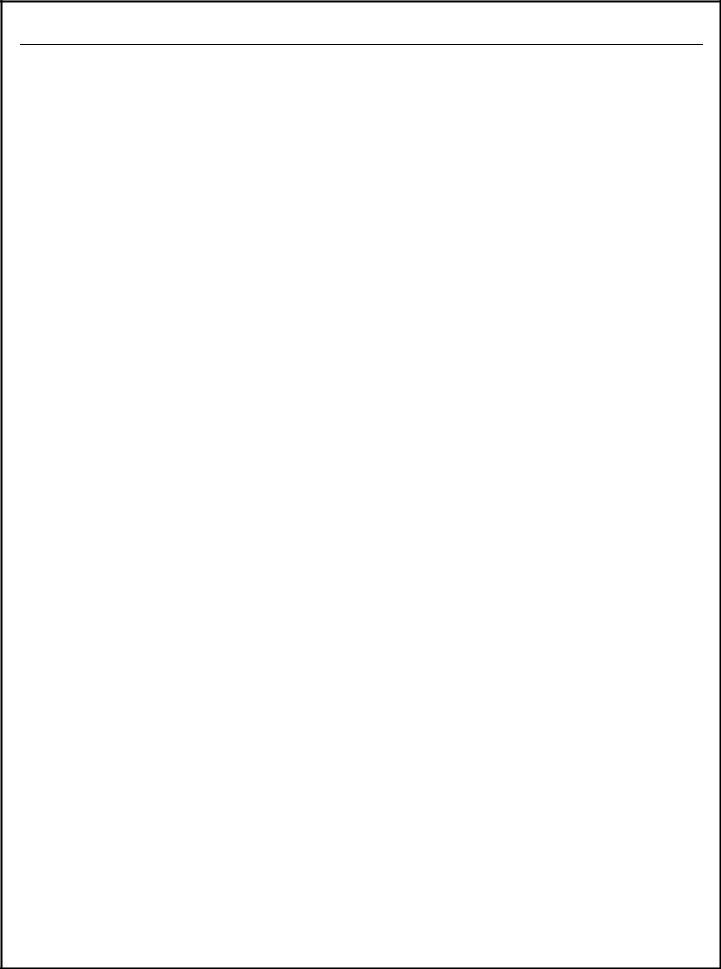
The algorithm ARoGS runs on each action rule generated by algorithm AR, and it produces the following additional action rule *(ar3)*:

*ar3 (d1 → d2) = (b, 2) ^ (a, 1 → 0) (d, 1 → 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.

*C. Association Acton Rules*

The strategy used earlier to find action rules earlier was 'lowest cost'. The Association–Action Rules (AAR) algorithm follows another approach in contrast to the one mentioned earlier. AAR makes use of Apriori algorithm for frequent action sets to generate association-type action rules.



**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 theinformation system supporting all attribute values in *actions*

*missingValues* ←set of missing flexibleattribute 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 theinformation 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

There is minimal attribute consideration in extracted action rules. The frequent action set generation is split 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 while in the pruning step: we do not take newly formed action set into consideration if it does not contain the decision action In our example, making use of the data from the Information System given in Table 1, the primary action sets generated by AAR are shown in Table 3. The frequent action sets generated by AAR are presented in Table 4. In our example, the action set is no longer useful if (d, 1, 2) is not present in it. Hence, the AAR algorithm generates frequent action sets and forms the association action rules from these action sets. Our put forward implementation of (Association Action Rule) AAR algorithm in a distributed environment is shown in Figure 5.

TABLE III. TABLE 3. PRIMARY ACTION SETS FOR INFORMATION SYSTEM S FROM TABLE 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Attribute** |  |  | **Primary action sets** |
|  | a |  |  | (a, a0), (a, a1), (a, a2), |
|  |  |  | (a, a0 → a1), (a, a0 → a2) |
|  |  |  |  |
|  |  |  |  | (a, a1 → a0), (a, a1 → a2), |
|  |  |  |  | (a, a2 → a0), (a, a2 → a1) |
|  | b |  |  | (b, b0) |
|  |  |  | (b, b2) |
|  |  |  |  |
|  | c |  |  | (c, c1 → c2) |
|  |  |  | (c, c2 → c1) |
|  |  |  |  |
|  | d |  |  | (d, d1 → d2) |
|  |  |  | (d, d2 → d1) |
|  |  |  |  |
|  | TABLE IV. | | TABLE 4. FREQUENT ACTION SETS FOR INFORMATION | |
|  |  |  |  | SYSTEM S FROM TABLE 1. |
|  |  | |  |  |
|  | **Iteration #** | |  | **Frequent action sets** |
|  | 1 |  |  | (a, a0) ^ (d, d1 → d2) |
|  |  |  | (a, a1) ^ (d, d1 → d2) |
|  |  |  |  |
|  |  |  |  | (a, a2) ^ (d, d1 → d2) |
|  |  |  |  | (b, b0) ^ (d, d1 → d2) |
|  |  |  |  | (b, b2) ^ (d, d1 → d2) |
|  |  |  |  | (a, a0 → a1) ^ (d, d1 → d2) |
|  |  |  |  | (a, a0 → a2) ^ (d, d1 → d2) |
|  |  |  |  | ……. |
|  |  |  |  | ……. |
|  | **Iteration #** | |  | **Frequent action sets** |
|  | 2 |  |  | (a, a0) ^ (b, b0) ^ (d, d1 → d2) |
|  |  |  | (a, a1) ^ (b, b0) ^ (d, d1 → d2) |
|  |  |  |  |
|  |  |  |  | (b, b0) ^ (c, c1) ^ (d, d1 → d2) |
|  |  |  |  | (a, a0 → a1) ^ (b, b0) ^ (d, d1 → d2) |
|  |  |  |  | ……. |
|  |  |  |  | ……. |
|  | **Iteration #** | |  | **Frequent action sets** |
|  | 3 |  |  | (a, a0) ^ (b, b0) ^ (c, c1) ^ (d, d1 → d2) |
|  |  |  | (a, a1) ^ (b, b0) ^ (c, c1) ^ (d, d1 → d2) |
|  |  |  |  |
|  |  |  |  | (a, a2) ^ (b, b0) ^ (c, c1) ^ (d, d1 → d2) |
|  |  |  |  | (a, a0) ^ (b, b0) ^ (c, c1 → c2) ^ (d, d1 → d2) |
|  |  |  |  | ……. |
|  |  |  |  | ……. |

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

*aar2 (d1 → d2) = (a, 1 → 0) (d, 1 → 2)*

*aar3 (d1 → d2) = (b, 0) ^ (a, 1 → 0)  (d, 1 → 2)*

*aar4 (d1 → d2) = (c, 0) ^ (a, 1 → 0) (d, 1 → 2) ……..*

*……..*

*aarn-1 (d1 → d2) = (b, 0) ^ (c, 0) ^ (a, 1 → 0) (d, 1 → 2)*

*aarn (d1 → d2) = (b, 0) ^ (a, 1 → 0) ^ (c, 2 → 1)  (d, 1 → 2)*

*D. Support and Confidence of Action Rules*

Consider an action rule *R* of the form (*Y1* → *Y2*) (*Z1* → *Z2*) where,

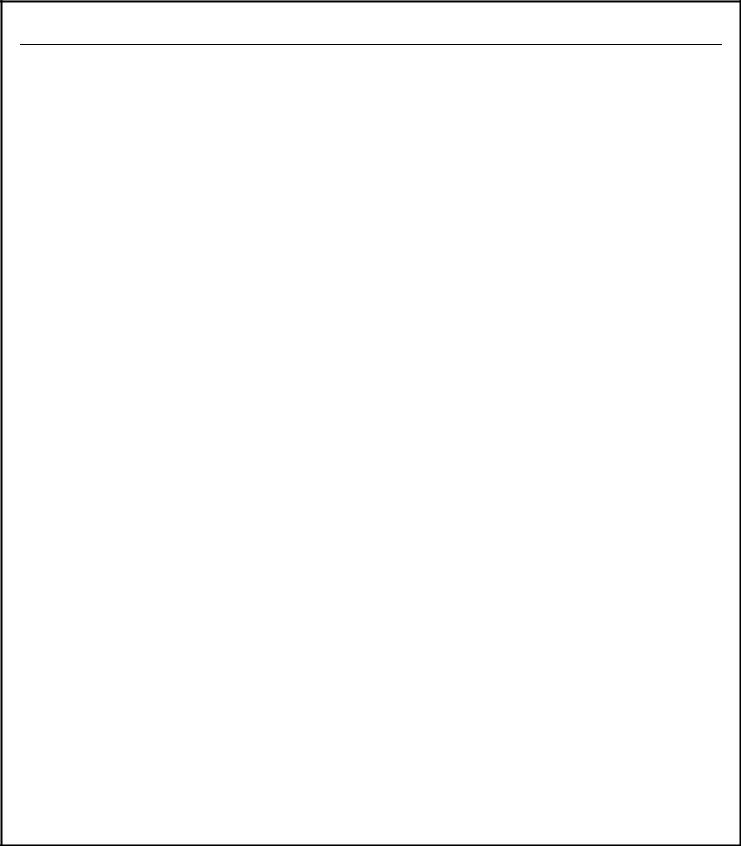
*Y* is concatenation of all action sets that support thedecision action *Z*

*Y1* = attribute values on left side of all actions in theleft side of the action rule *R*

*Y2* = attribute values on right side of all actions inthe left side of the action rule *R*

Z1 = decision attribute value on the left side

Z2 = decision attribute value on the right side



**ALGORITHM 4:**

AAR

*primaryActionSets* ← a set of all attribute values. If

an attribute is flexible, all possible combinations of

atomic action sets of that attribute is also added

*tempActionSets* ← *primaryActionSets*

*newActionSets* ← empty list to store new combination of action set

**while** *tempActionSets*is **not empty do**

**for each** *set1*in*tempActionSets* **do**

**for each** *set2*in*primaryActionSets* **do if** (*set1*contains decisionAction **or** *set2* contains decisionAction)**and**attribute of *set2* is not a subset of attributes of *set1* **then**

*newActionSets*. Add(combinedlist of *set1* and *set2*)

**end**

**end**

**end**

**Output** each action set in*newActionSets*asan action rule

*tempActionSets* ← *newActionSets*

**end**

Figure 5. AAR (Association Action Rules) algorithm in distributed environment using MapReduce

*aar1 (d1 → d2) = (a, 2 → 0) (d, 1 → 2)*

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 [card (Y1^ Z1), card (*Y2 ^ Z2*)] *Confidence (aar) =* [*card (Y1 ^ Z1) / card (Y1)*] \*[*card (Y2* *^ Z2) / card (Y2)*]

where *card(Y1)* ≠ 0 and *card(Y2)* ≠ 0

1. *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 (Y2 ^ Z2)*

*Old Confidence (ar) =* [*card (Y1 ^ Z1) / card (Y1)*] \*[*card (Y2* *^ Z2) / card (Y2)*]

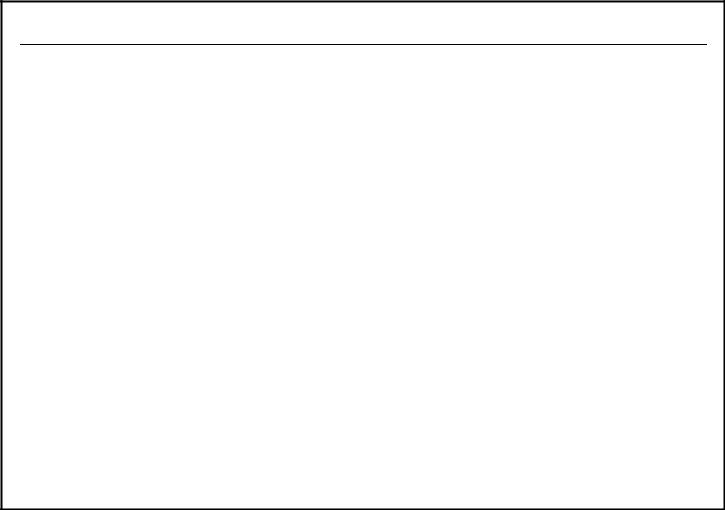
In our proposed method, this confidence is replaced by the following confidence formula given by Tzacheva et al. [17] to reduce complexity:

*New Confidence (ar) =* [*card (Y2 ^ Z2) / card (Y2)*]where *card(Y1)* ≠ 0 and *card(Y2)* ≠ 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 ar3 (d1 → d2) and aar3 (d1 → d2), 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.

E. 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 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.



**ALGORITHM 5:** Reduce (*Key*,*values*)

(where Key is an action rule from Algorithm *AR* or Algorithm *AAR* and values is a list ofsupport 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

**IV. EXPERIMENT AND RESULTS**

We used one dataset for testing our proposed MR - Random-Forest algorithm for distributed action rules discovery: Car Evaluation 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 for this study, in order to illustrate the application of Action Rules in defining safety of the people .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.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:

*aar1(label,labelunacc -> labelacc)^(buying,buyingmed -> buyingmed)^(persons,personsmore -> persons4)^(lug\_boot,lug\_bootbig -> lug\_bootsmall)^(maint, -> mainthigh) ==> (safety,safetylow -> safetyhigh) 2 [Support: 2.0, Confidence: 100.0%]*

The rule *arCar1* means that: if the buying price of the car remains medium (*buyingmedium*), and the number of persons it can carry increases from 2 (*persons2*) to 4 (*persons4*), and the label of the car changes from *lableunacc* to *labellacc*, then the decision attribute (*class*) value is expected to change from low(safety) to high (*saftey*). 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 *{ Doors}* are designated as *Stable Attributes*, and the rest of attributesare designated as *Flexible Attributes* exluding theattribute *Saftey* which is designated as the *decision attribute*, which is also a flexible attribute. These parameters are shown in Table 6.

|  |
| --- |
| TABLE V. |
|  |
|  |  |  | |  | |  | |
| **Property** |  |  | | **Car** | | **Evaluation** | |
|  |  |  | | **Dataset** | | | |
| Number of | |  | |  | | 1728 | |
| instances |  |  | |  | |  | |
| Attributes | |  | | 7 attributes | | | |
|  |  |  | | • Buying | | | |
|  |  |  | | • Maintenance | | | |
|  |  |  | | • Doors | | | |
|  |  |  | | • Persons | | | |
|  |  |  | | • Luggage boot | | | |
|  |  |  | | • Safety | | | |
|  |  |  | | • Class | | | |
| Decision |  |  | |  | |  | |
| attribute values | |  | |  | | Saftey | |
|  |  |  | |  | | | |
| Original data | |  | | 52.3 Kilo Bytes | | | |
| size |  |  | |  | |  | |
| TABLE VI. | | |
|  | | |  | | MASS DATASET | |
|  | | |  | |  | |
| **Parameters** | | |  | | **Car** | |
|  | | |  | | **Evaluation** | |
|  | | |  | | **Dataset** | |
| Stable | | |  | | Doors | |
| attributes | | |  | |  | |
|  | | |  | |  | |
| Expected | | |  | | (Class) | |
| decision action | | |  | | Saftey low to high | |
| Minimum | | |  | | 150, 80% | |
| support and | | |  | |  | |
| confidence | | |  | |  | |

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. 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.

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: ‘*re-classify* *a breast tumor form malignant to benign severity*’; inmedical domain: ‘*change the car state from unacceptable* *to acceptable*’. In other domains example goals can be: infinancial domain: ‘*increase the customer loyalty*’; ‘*how to decrease the risk of a loan*’; education domain: ‘*how to improve student evaluations*’.

TABLE VIII. TABLE 8. COMPARISON OF GENERAL AND SPECIFIC ACTION RULES PRODUCED BY AROGS AND AAR RESPECTIVELY

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

Considering the fact thatnowadays 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.

**V. 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, 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, 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, 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,

160-168.

1. Z.W. Ras, A. Dardzinska (2006), Action rules discovery, a new simplified strategy, Foundations of Intelligent Systems, LNAI, No. 4203, Springer, 445-453.
2. 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, 293-300.
3. 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, 95-105.
4. 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, 322-329.
5. 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, 772-776.
6. 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, 283-290.
7. 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.
8. 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.

1. L. Breman (2001), Random Forests, in Machine Learning, Vol. 45, Kluwer Academic, 2001, pp. 5-32
2. X. Lin, (2014), MR-Apriori: Association Rules algorithm based on MapReduce, in 5th IEE International Conference on Software Engineering and Service Science (ICSESS) 2014, Beijing, China, pp. 141-144.
3. 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.
4. M. Lichman (2013), UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.
5. A.A. Tzacheva, C.C. Sankar, S. Ramachandran, R.A. Shankar (2016), Support Confidence and Utility of Action Rules Triggered by Meta-Actions, unpublished.
6. 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.