Data Distribution Method for Scalable Actionable Pattern Mining

Arunkumar Bagavathi
University of North Carolina at Charlotte
Charlotte, NC
abagavat@uncc.edu

Varun Rao
University of North Carolina at Charlotte
Charlotte, NC
vrao3@uncc.edu

Angelina A. Tzacheva
University of North Carolina at Charlotte
Charlotte, NC
aatzache@uncc.edu

ABSTRACT

Action Rules are rule based systems for discovering actionable patterns hidden in a large dataset. Action Rules recommend actions that a user or a system can undertake to their advantage, or to accomplish their goal. Current Action Rules extraction methods are unable to process huge volumes of data in a reasonable time and it requires a distributed and parallel extraction methods. Limited research has been done on extracting Action Rules using a distributed scenario. Major complications of discovering Action Rules with such distributed systems are data distribution among computing nodes and calculation of major parameters of action rules. In this work, we propose few methods to handle the big data distribution among computation nodes using the Spark framework. With enhanced experiments made on datasets in transportation, medical, and business domains, we show our methods achieve almost equal valid results compared to results from classical non-distributed Action Rule discovery methods with improved run time.

KEYWORDS

Action Rules, Data Distribution, Distributed Processing

ACM Reference Format:
Arunkumar Bagavathi, Varun Rao, and Angelina A. Tzacheva. 2018. Data Distribution Method for Scalable Actionable Pattern Mining. In *Proceedings of* . ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Knowledge discovery is a process of preprocessing, transforming and identifying unidentified patterns and trends from a large quantity data. Rule based knowledge discovery tasks intend to circumscribe methods that identifies, learns or evolves 'rules' to store and manipulate knowledge. Rule based systems are used for finding association and classifying data objects. Rules takes the format as given in equation (1), where the *antecedent*(left side of the rule) is a conjunction of conditions and the *consequent* (right side of the rule) is a resulting pattern for the conditions in antecedent.

$$condition(s) \rightarrow result(s) \tag{1}$$

Action Rule is also rule based knowledge discovery technique that recommend possible transitions of data from one state to another, which the user can use to their advantage. For example, one would want to find actionable patterns in the data to improve his/her salary. Some of the applications for Action Rules are: improving customer satisfaction in business [9] and reducing hospital readmission in the medical field [3]. Action Rules are extracted from Decision table [15]. In decision table, attributes can be split into *Stable Attributes* and *Flexible Attributes* along with the *Decision attribute* which is the final decision that the user need to achieve. Stable attributes in any Action Rule *AR* remain constant or cannot form action in *AR*. While flexible attributes can change their value from a_i to a_j. Action Rules can take the representation as given in equation (2), where Ψ represents a conjunction of stable features, $(\alpha \rightarrow \beta)$ represents a conjunction of changes in values of flexible features and $(\theta \rightarrow \phi)$ represents desired decision action.

$$[(\Psi) \wedge (\alpha \rightarrow \beta)] \rightarrow (\theta \rightarrow \phi) \tag{2}$$

More than a decade, many researches have been conducted over Action Rules giving rise to several algorithms like DEAR [19], ARAS [16] and Association Action Rules [14]. These algorithms extract Action Rules in an expected time frame when the dataset size is limited, which is not the case these days. Limited research has been done on extracting Action Rules in a distributed scenario. Some methods like MR-Random Forest[20] method has been proposed to introduce the concept of distributed Action Rule discovery. Challenges in extracting Action Rules in a distributed fashion include distributing the data among nodes and combining results from all nodes such that there is no loss of any patterns from the data, we get Action Rules in a fair amount of time and there is less communication overhead for the cluster. In this paper, we propose a method to handle the data distribution task and extract Action Rules efficiently using the Spark framework. Although we propose this method for the data that contains many attributes, we show that the proposed method suits also for other datasets. We also apply our algorithm to datasets in the domains of transportation, medical and business.

2 RELATED WORKS

The perception of Action Rules is first introduced in 2000, when Ras and Alicia proposed an idea of Action Rules help many businesses to gain profit by finding interesting actionable patterns in the data [15]. In the literature, Action Rules are extracted using two methods. First is a rule based approach, in which intermediate classification rules are extracted first using efficient rule generation algorithms such as LERS or ERID. From these extracted rules, action rules are generated with systems like DEAR [19], which extracts Action Rules from two classification rules, or ARAS [16], which extracts Action Rules using a single classification rule. Second method is object-based approaches, in which the Action Rules are extracted

, ,
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Table 1: Sample Decision System S

X	a	b	c	d
x_1	a_1	b_1	c_1	d_1
x_2	a_3	b_1	c_1	d_1
x_3	a_2	b_2	c_1	d_2
x_4	a_2	b_2	c_2	d_2
x_5	a_2	b_1	c_1	d_1
x_6	a_2	b_2	c_1	d_2
x_7	a_2	b_1	c_2	d_2
x_8	a_1	b_2	c_2	d_1

directly from the decision table without any intermediary steps. Systems ARED [7] and Association Action Rules [14] works in the object-based approach. Out of all algorithms, only Association Action Rules [14] extracts all possible Action Rules from the given decision table but it is inefficient in terms of time to extract rules.

Considering the growth of big data, some research [20] [4] has been done to extract Action Rules using popular distributed computing frameworks such as MapReduce and Spark. The main challenge involved in distributed processing of rule based data mining is load balancing and obtaining global optimum results. [20] proposed a method to distribute the data randomly to different sites and gather results from all sites. [4] handle the load balancing by clustering the data into d partitions, where d is the number of unique values of a decision attribute. Since not much of the works have been done on distributed Action Rule algorithms area, we give related works data load balancing for other problems such as association rule mining.

Apriori algorithm is one of the popular algorithms to find frequent patterns [2]. Since the algorithm has to find all combination of patterns from the data and it involves many passes over the data, several parallel algorithms have been proposed to discover frequent itemsets faster. Over a decade many research has been roposed to extract association rules in a distributed setup [11][21] [13] [17] [12] [18] [22].

In this paper, we propose a novel approach for partitioning the given data. We also give a new algorithm to extract all Action Rules, based on the algorithm proposed in [14], which is the slowest among all proposed Action Rules algorithms and compute additional parameters like Utility and Coverage. We test how fast our new method works compared to our previous distributed Action Rule extraction algorithms and also we check validity of the new method by comparing number of Action Rules generated and rule coverage of Action Rules from system with classical Association Action Rules [14] on a single machine and SARGS [4] systems.

3 ACTION RULES AND SPARK

In this section, we give basic knowledge about Decision system, Action Rules, Spark and GraphX frameworks to understand out methodology.

3.1 Action Rules

In this subsection, we give definitions of action terms, action rules and properties of action rules [15]

Let $S = (X, A \cup d, V)$ be a decision system, similar to the one given in Table 1, where d is a decision attribute and $V = \cup V_i : i \in A$. Action terms can be given by the expression of $(m, m_1 \rightarrow m_2)$, where $m \in A$ and $m_1, m_2 \in V_m$. $m_1 = m_2$ if $m \in A_{St}$. In that case, we can simplify the expression as (m, m_1) or $(m = m_1)$. Whereas, $m_1 \neq m_2$ if $m \in A_{Fl}$

Action Rules can take a form of $t_1 \cap t_2 \cap .... \cap t_n$, where t_i is an atomic action or action term and the Action Rule is a conjunction of action terms to achieve the desired action based on attribute d. Example Action Rule for the Decision System in *Table 1* is given below: $(a, a_1 \rightarrow a_2).(b, b_1 \rightarrow b_2) \longrightarrow (d, d_1 \rightarrow d_2)$

3.1.1 Properties of Action Rules. Action Rules are considered interesting based on the metrics such as Support, Confidence, Utility and Coverage. Higher these values, more interesting they are to the end user.

Consider an action rule R of form:
$(Y_1 \rightarrow Y_2) \longrightarrow (Z_1 \rightarrow Z_2)$ where,
Y is the condition part of R
Z is the decision part of R
Y_1 is a set of all left side action terms in the condition part of R
Y_2 is a set of all right side action terms in the condition part of R
Z_1 is the decision attribute value on left side
Z_2 is the decision attribute value on right side
In [15], the support and confidence of an action rule R is given as

$Support(R) = min\{card(Y_1 \cap Z_1), card(Y_2 \cap Z_2)\}$

$Confidence(R) = [\frac{card(Y_1 \cap Z_1)}{card(Y_1)}] \cdot [\frac{card(Y_2 \cap Z_2)}{card(Y_2)}]$

Later, Tzacheva et.al [1] proposed a new set of formula for calculating Support and Confidence of Action Rules. Their idea is to reduce complexities in searching the data several times for Support and Confidence of an Action Rule. The new formula are given below.

$Support(R) = \{card(Y_2 \cap Z_2)$

$Confidence(R) = [\frac{card(Y_2 \cap Z_2)}{card(Y_2)}]$

Tzacheva et. al [1] also introduced a notion of utility for Action Rules. Utility of Action Rules takes a following form. For most of cases Utility of Action Rules equals the Old Confidence of the same Action Rule.

$Utility(R) = [\frac{card(Y_1 \cap Z_1)}{card(Y_1)}]$

Coverage of an Action Rule means that how many decision from values, from the entire decision system S, are being covered by all extracted Action Rules. In other words, using the extracted Action Rules, *Coverage* defines how many data records in the decision system can successfully transfers from Z_1 to Z_2.

3.2 Spark

Spark [23] is a framework that is similar to MapReduce [5] to process large quantity of data efficiently in a parallel fashion and in a short span of time. The disadvantage of MapReduce framework is frequent system's disk access for writing and reading the data between Map and Reduce phases. However, Spark introduces a distributed memory abstraction strategy named Resilient Distributed Datasets(RDD). The RDDs works by splitting the data into multiple nodes, do in-memory computations on whose nodes and store the results in memory itself if there are any available space in RAM. These results can be accessed for future processes and analyses,

which in-turn create another RDD. Thus, Spark cuts-off large number of disk accesses for storing intermediate outputs like in Hadoop MapReduce.

Spark helps machine learning algorithms which relies on multiple iterations on the given data with the help of RDD's in-memory computation. Spark handles node failures by having a lineage graph of RDDs. The lineage graph is a Directed Acyclic Graph(DAG) where each node represents a transformation stage. When a failure occurs at a certain stage, Spark uses the last available working point(RDD) from the lineage graph and restart all computations from that working point rather than repeating the entire process from the beginning or saving the intermediate results and replicating them across multiple nodes.

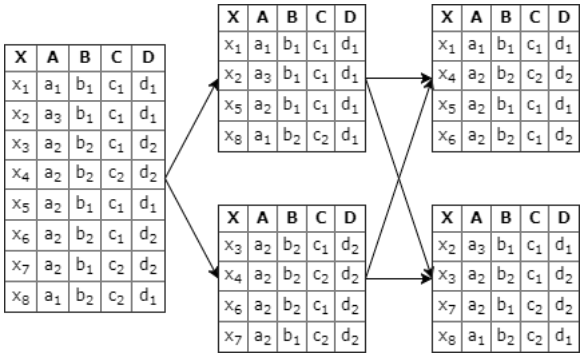

Figure 1: Example Data Distribution in SARGS for Table 1

4 METHODOLOGY

In this paper, we propose a new method for generating Action Rules in a distributed fashion. For scalability, we redesign the Action Rules algorithms and implement them in the Spark framework [23]. The basic problem in previous works [20] [4] related to Action Rules is the data distribution module. They divided the data by random by using the automatic partitioning with Apache Spark and MapReduce We propose two new methods for more intelligent scalable data distribution: (1) Slitting data based on decision attribute values distribution, (2) vertically splitting the data. We compare our results with the results from other distributed Action Rule extraction : SARGS [4].

4.1 Method 1. Class Attribute Value - Data Distribution Strategy for Scalable Action Rules

4.1.1 Data Distribution Module. The data distribution module is to evenly distribute the data based on the decision attribute. The main objective of the data distribution module is to overcome the obstacle of inaccurate knowledge discovery while extracted in a distributed setup. The given input data is split into n groups, where *n=no. of decision attribute vales* and each group consists of records from the information system matching the corresponding decision value. Also, the proportion constraint $P_g \simeq P_S$ is maintained, where P_g is the proportion of records in a partition g with decision attribute value d_i and P_S is the proportion of records in the given information system S with decision attribute value d_i. By this way, each partition contains same proportion of data which is equal to the original dataset. The final actionable knowledge from these partitions are considered to be equal to that of the knowledge from the single data. Figure 1 shows an example data partition for the information system S shown in Table 1.

SARGS algorithm consists of 3 modules namely: Data distribution, LERS and ARAS.

4.1.2 LERS module. After the data is distributed, we extract Action Rules on each patition by using a set of 2 classification rules produced by Learning fro Example based on Rough Sets (LERS) system [8]. The second module does the LERS [8] classification rule induction. 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. Since LERS follows bottom-up strategy, it constructs classification rules with conditional part covering x attributes, then it continues to construct rules with conditional part of $x + 1$ attributes during the following iterations. Only marked rules from the LERS module are considered for the ARAS module. A classification rule c_i if and only if $S_{c_i} \subseteq S_{d_*}$, where S_{c_i} is the set of rows in S that support the classification rule c_i and S_{d_*} is the set of rows in S that support the decision attribute value d_*.

4.1.3 SARGS - Modified ARAS module. The SARGS algorithm propsed in [4] uses LERS [8] and ARAS [16] methods for extracting Action Rules in a distributed fashion for larger datasets.

The third module uses SARGS [4] - the modified version of ARAS [16] and it uses all marked classification rules from the second (LERS) module and derives Action Rules. ARAS method, which extracts incomplete Action Rules, may not be useful when the user requires valid recommendations. Sample Action Rules from the system ARAS for the Decision System S given in Table 1 are given below:

$ARs_1 : (d_1 \rightarrow d_2) = (a, \rightarrow a_2).(b, \rightarrow b_2) \longrightarrow (d, d_1 \rightarrow d_2)$
$ARs_2 : (d_1 \rightarrow d_2) = (a, \rightarrow a_2).(c, c_2) \longrightarrow (d, d_1 \rightarrow d_2)$
$ARs_3 : (d_1 \rightarrow d_2) = (b, \rightarrow b_1).(c, c_2) \longrightarrow (d, d_1 \rightarrow d_2)$
$ARs_4 : (d_1 \rightarrow d_2) = (b, \rightarrow b_2).(c, c_1) \longrightarrow (d, d_1 \rightarrow d_2)$

Algorithm 1 SARGS

Require: **actions** of type list(actions) and d_{FROM} values
1: **procedure** PROCEDURE
2: $s_v \leftarrow$ list of stable attribute values in *actions*
3: $a_s \leftarrow$ set of objects in decision system supporting $s_v \cap d_{FROM}$
4: $m_v \leftarrow$ set of missing flexible attribute values in *actions*
5: $s_{mv} \leftarrow$ Cartesian product of missing values
for each *valueSet* in c_{mv} **do**:
6: $n_v \leftarrow$ combine valueSet with s_v
7: $n_s \leftarrow$ set of objects in the decision system supporting n_v in *actions*
8: **If** $n_s \subseteq a_s$ **then**
9: Add *value* to *actions*
10: **Output** *actions* as Action Rule
11:

Algorithm 1 gives the modified version of ARAS module that the SARGS algorithm uses to extract all complete Action Rules. This algorithm extracts all missing values from the conditional (left) part of the given Action Rule. The algorithm then get cartesian product of all missing values (except the values of same attribute) and fills in the action rule. Following Action Rules are extracted from the decision system S given in Table 1 using SARGS method.

$AR_1(d_1 \rightarrow d_2) = (A, a_1 \rightarrow a_2).(B, \rightarrow b_2) \longrightarrow (D, d_1 \rightarrow d_2)$

$AR_2(d_1 \rightarrow d_2) = (A, a_3 \rightarrow a_2).(B, \rightarrow b_2) \longrightarrow (D, d_1 \rightarrow d_2)$

4.2 Method 2. Vertical Split - Data Distribution for Scalable Association Action Rules

In this work, we propose a novel approach for extracting Action Rules by splitting the data vertically, in cotrast with the classical horizontal split, which is performed by parallel processing systems. This vertical split method can be applied to the Association Action Rules discovery method described by Ras et. al [14]. Association Action Rules is an exhaustive A-Priori like method, which extracts all possible action rules by taking all combinations of Action terms through iterative nature. For that reason, Association Action Rules is the most complex, and the most computationally expensive out of all Action Rules extraction algorithms. In this work, we propose the Vertical Data Split method, which allows for much faster computational time for Association Action Rules extraction, as well as it it makes it possible to run the extraction in a Cloud Cluster environment in parallel.

$(a, a_1) \cap (b, b_2) \longrightarrow (c, c_1) \cap (d, d_2)$

We propose a method provides very broad recommendations and works comparatively faster than our previous approaches: MR-Random Forest algorithm [20] and SARGS algorithm [4]. We propose this method that suits data that has large attribute space. But the method works better with small data also. In this method, we split the data vertically into 2 or more partitions, with each partition having only a small subset of attributes. Our algorithm runs separately on each partition, does transformations like *map(), flatmap()* functions and combine results with *join()* and *groupBy()* operations. Algorithm 2 gives our new algorithm to extract all possible Action Rules from the data in a parallel fashion. Following this algorithm for each partition of data, we collect multiple action rules set. We combine all action rules to give user a final set of recommendation action rules.

Algorithm 2 ActionRulesExtract

Require: **data** of type (r_{id}, r_{values})) and d_{FROM}, d_{TO} values
  procedure MyProcedure
    $d_A := (s \in r | r \in data | (s, r_{id})).groupByKey()$
    $c_{OLD} \leftarrow d_A$
    $i \leftarrow 2$
  parallel:
    while i ≠ n **do**:
      $c \leftarrow data.flatmap(r => (comb(r_{values}, i)), r_{id})$
      $c_{NEW} \leftarrow c.groupByKey()$
      $c_{VALID} \leftarrow c_{NEW}.filter()$
      $c_{FROM} \leftarrow c_{VALID}.filter(d_{FROM})$
      $c_{TO} \leftarrow c_{VALID}.filter(d_{TO})$
      if $c_{FROM} = \emptyset$ or $c_{TO} = \emptyset$ **then break**
      $atomic \leftarrow c_{FROM}.jon(c_{TO}).filter()$
      $action_{supp} \leftarrow (r \in atomic \mid findSupp(r)).filter()$
      if $action_{supp} = \emptyset$ **then break**
      $atomic_{FROM} \leftarrow atomic.filter()$
      $atomic_{TO} \leftarrow atomic.filter()$
      $a_{FROM} \leftarrow atomic_{FROM}.join(c_{OLD})$
      $a_{TO} \leftarrow atomic_{TO}.join(c_{OLD})$
      $action_{conf} \leftarrow a_{FROM}.join(a_{TO})$
      $actions \leftarrow action_{supp}.join(action_{conf})$
      collect *actions*
      $c_{OLD} := c_{NEW}$
      $i := i + 1$

5 EXPERIMENTS AND RESULTS

To test our methods, we use three datasets: *Car Evaluation* data, *Mammographic Mass* data, and the Charlotte North Carolina BusinesWise data.

The Car Evaluation and Mammography are obtained from the Machine Learning repository of the Department of Information and Computer Science of the University of California, Irvine [10]. The Car Evaluation Data consists of records describing a car's goodness and acceptability. The Mammographic Mass data contains records that measure severity of the cancer. Since these datasets are relatively small in size, in order to test them for scalability with the proposed distributed processing algorithms, we replicate their data rows 1024 and 2056 times respectively for CarEvaluation and MammographicMass datasets, in order to increase data size.

We also test with the city of Charlotte North Carolina BusinesWise data , which is donated by the Charlotte Chamber of Commerce. This data collects details of over 20,000 business companies in Mecklenburg county, North Carolina. From this data, our focus is how to increase a company's estimated sales from <2 million US dollars to the range between 3 million and 10 million US dollars. Further Table 2 gives a broad picture of the datasets that we used to test our algorithm.

Table 3 show parameters that we set for each dataset to collect Action Rules. For our vertical data distribution method evaluations, we split the Business data only because of the small number of attributes in Car Evaluation data and Mammographic Mass data.

In Table 4, we give the performance of Class data distribution methods in context of number of Action Rules they extract from each dataset. From Table 4, it can be noted that the *SARGS* algorithms, using both default data distribution and class distribution approaches, fail to extract some Action Rules. This is due to the data distribution step involved before the algorithm start to extract Action Rules. It can be noted that for the Business data, we get 142 rules in common from 265 rules. The remaining rules are based on the data distribution of different partitions.

In Table 5, we give performance of Vertical data distribution method for Association Action Rules [14] in terms of number of Action Rules extracted and compare them with the same algorithm in non-parallel method. From Table 4 and Table 5, it can be noted that Association Action Rules method extracts more number of Action Rules compared to the SARGS method. This is due to the

Table 2: Dataset Properties

Property	Car Evaluation Data	Mamm. Mass Data	Business Data
Attributes	7 attributes -Buying - Maintenance -Doors -Persons -Luggage Boot -Safety -Class	6 attributes -BI-RADS -Patient's age -Shape -Margin -Density -Severity	17 attributes including -City -Sector -Site Type -Building Type -Estimated Sales -Total Employees Count
Decision attribute values	Class (unacc, acc, good, vgood)	Severity (0 - benign, 1-malignant)	Estimated Sales ($< \$2M$, $3-10M$, $10-25M$, $25-50M$, $50-100M$, $100-500M$, $> 500M$)
# of instances / decision value	unacc - 1210 acc - 384 good - 69 vgood - 65	0 - 516 1 - 445	$< \$3M$ - 12503 \$3-\$10M - 1927 \$10-\$25M - 393 \$25-\$50M - 130 \$50-\$100M - 69 \$100-\$500M - 57 $> \$500M$ - 50
# of instances after replication	1,769,472	1,968,128	N/A

fact that the Association Action Rules method extract all possible Action Rules from the data.

In Table 6 and Table 7, we give run time evaluations of our proposed methods. Table 6 compares runtimes of SARGS methods with the classic ARAS [16] method. From this table, it can be noted that the SARGS method [4] with default data distribution performs better, in terms of run time, than our proposed method for all datasets. This is due to an extract data distribution step involved in the Class data distribution method.

In Table 7, we give run time evaluations of our proposed method 2: vertical data distribution for the Association Action Rules [16] method. Association Action Rules extraction is one of the complex

Table 3: Parameters used in all Action Rule discovery algorithms

Property	Car Evaluation Data	Mamm. Mass Data	Business Data
Stable attributes	Maintenance, Buying Price, Doors	Age, Shape	Start Year
Required decision action	(Class) $unacc \rightarrow acc$	(Severity) $1 \rightarrow 0$	Estimated Sales $\$3M - \$10M \rightarrow \$10M - \$25M$
Minimum Support α and Confidence β	2048, 70%	4096, 70%	10, 70%

Table 4: Performance of SARGS algorithms with Class Data Distribution in terms of number of rules generated. Values in brackets() represent number of common Action Rules across all methods

Data	Non-Parallel Algorithm	SARGS - Default Data Distribution	SARGS - Class Distribution Algorithm
Car Evaluation Data	53	53	53
Mamm. Mass Data	166	165	165
Business Data	265	372(142)	418(142)

Table 5: Performance of Association Action Rules algorithm with Vetical data distribution in terms of number of rules generated

Data	Non-Parallel Algorithm	Vertical Data Distribution Algorithm
Car Evaluation Data	3500	3496
Mamm. Mass Data	5790	5756
Business Data	67000	66632

Action Rule technique because the algorithm needs to evaluate all possible combination of action terms. With our additional vertical data distribution method to Spark's default data distribution, we achieve better performance in terms of run time of the algorithm.

Table 6: Time taken by SARGS algorithms with data distribution methods to extract Action Rules

Data	Non-Parallel Algorithm	SARGS - Default Data Distribution	SARGS - Class Distribution
Car Evaluation Data	0.56 mins	0.28 mins	0.5 mins
Mamm. Mass Data	0.75 mins	0.38 mins	0.51 mins
Business Data	20 mins	13.4 mins	14 mins

Table 7: Time taken by Association Action Rules algorithms to extract Action Rules

Data	Non-Parallel Algorithm	Vertical Data Distribution method
Car Evaluation Data	72 mins	1.73 mins
Mamm. Mass Data	40 mins	1.45 mins
Business Data	$> 20hrs$	22 mins

In Table 8, we give sample Action Rules, that are common in all algorithms, for the Car Evaluation data ($AR_{C1}, AR_{C2}, AR_{C3}$), Mammographic Mass data ($AR_{M1}, AR_{M2}, AR_{M3}$) and the Business data ($AR_{B1}, AR_{B2}, AR_{B3}$). For example, consider the Action Rule AR_{B1}. This rule recommends that if a company moves from 'Residential' to 'Office' type and if they move from the city of 'Matthews' to 'Charlotte' and increase their overall employees count to the range of '$25 - 49Employees$' and if they change their sector to 'Services' and if they increase their number of offices and if their start year is between 1981 and 1990, their Estimated sales would increase from the range $\$2M - \$10M$ to the range $\$10M - \$24M$.

We now give evaluations on Action Rules based on classic single machine methods with our proposed methods. As described in Table 4, due to data distribution part in SARGS methods, we lose some Action Rules with our proposed methods. On the other hand, in the real world scenario, all Action Rules are not necessary to make a decision action. In Table 9 and Table 10, we compare properties of Action Rules such as *Support,Old Confidence, New Confidence and Utility* from our proposed class distribution with SARGS method with default data distribution and class data distribution methods for 3 different datasets. For space constraints, we limit our evaluations to the Action Rules given in Table 8 alone. We run all our algorithms in 10 partitions for Car Evaluation and Mammographic Mass datasets and 2 partitions for the Business data.

Table 8: Action Rules of all Datasets

Car Evaluation Data

(1) AR_{C1} : $(buying = low) \wedge (maint = vhigh) \wedge (persons, more \rightarrow 4) \Longrightarrow (class, unacc \rightarrow acc)$ [Support: 3072, Old Confidence: 70%, New Confidence: 100%, Utility: 70%]

(2) AR_{C2} : $(buying = med) \wedge (doors = 3) \wedge (maint = med) \wedge (persons, 2 \rightarrow more) \wedge (safety, high \rightarrow med) \Longrightarrow (class, unacc \rightarrow acc)$ [Support: 3072, Old Confidence: 100%, New Confidence: 100%, Utility: 100%]

(3) AR_{C3} : $(buying = med) \wedge (lugboot, small \rightarrow big) \wedge (maint = med) \wedge (persons, 2 \rightarrow more) \wedge (safety, low \rightarrow med) \Longrightarrow (class, unacc \rightarrow acc)$ [Support: 4096, Old Confidence: 100%, New Confidence: 100%, Utility: 100%]

Business Data

(1) AR_{B1} : $(BLDGTYPE, Residential \rightarrow Office) \wedge (CITY, Matthews \rightarrow Charlotte) \wedge (EMPALLSITE, 1 - 3Employees \rightarrow 25 - 49Employees) \wedge (SECTOR, AgricultureForestryandFishing \rightarrow Services) \wedge (SITETYPE, SingleSite \rightarrow Headquaters) \wedge (STARTYR, 1981 - 1990) \Longrightarrow (ESTSALES, \$2Mto\$10M \rightarrow \$10Mto\$25M)$ [Support: 11, Old Confidence: 73%, New Confidence: 73%, Utility: 73%]

(2) AR_{B2} : $(BLDGTYPE, Office/Retail \rightarrow Retail) \wedge (EMPALLSITE, 10 - 24Employees \rightarrow 50 - 99Employees) \wedge (SECTOR, RetailTrade \rightarrow Services) \Longrightarrow (ESTSALES, \$2Mto\$10M \rightarrow \$10Mto\$25M)$ [Support: 12, Old Confidence: 100%, New Confidence: 100%, Utility: 100%]

(3) AR_{B3} : $(CITY, Matthews \rightarrow Huntersville) \wedge (EMPALLSITE, 25 - 49Employees \rightarrow 100 - 249Employees) \wedge (SITETYPE, SingleSite \rightarrow Headquarters) \Longrightarrow (ESTSALES, \$2Mto\$10M \rightarrow \$10Mto\$25M)$ [Support: 4, Old Confidence: 64%, New Confidence: 100%, Utility: 64%]

6 CONCLUSION

We propose two new methods for Data Distribution for Cloud Parallel processing , which can be applied to Actionable Pattern Mining via Action Rules. The Method 1 can be applied to most Action Rules algorithms, including ActionRules [15], system DEAR [19], ARAS [16], systems ARED [7].

Method 2 is specifically designed for Association Action rules [14], which is the most complex and time-consuming Action Rules extraction method, however it discovers all possible Action Rules. Previous works divide the data by random using default partitioning provided by Hadoop MapReduce, and Apache Spark. For that reason, the calculation of Support and Confidence may not represent very well the original support and confidence for Action Rules

Table 9: Support(s),Old Confidence(o),New Confidence(c) and Utility(u) of Action Rules from Car Evaluation for the Class Distribution approach

Action Rule	Non-parallel algorithm				SARGS - Default Data Distribution				SARGS - Class Distribution			
	s	o	c	u	s	o	c	u	s	o	c	u
AR_{C1}	3072	70%	100%	70%	343	42%	100%	42%	339	70%	100%	70%
AR_{C2}	3072	100%	100%	100%	306	100%	100%	100%	309	100%	100%	100%

Table 10: Support(s),Old Confidence(o),New Confidence(c) and Utility(u) of Action Rules from Business Data for the Class Distribution approach

Action Rule	Non-parallel algorithm				SARGS - Default Data Distribution				SARGS - Class Distribution			
	s	o	c	u	s	o	c	u	s	o	c	u
AR_{B1}	11	73%	73%	73%	4	100%	100%	100%	7	66%	66%	66%
AR_{B2}	12	100%	100%	100%	5	0%	0%	0%	7	100%	100%	100%

extracted on the entire dataset, or the support and confidence may be incorrect all together.

Our results show improved support, confidence, and utility with the new proposed methods, which more closely represent the correct support and confidence as obtained by non-parallel methods. In addition, our results show much faster computational times with big datasets for the exhaustive Association Action Rules method. Future work includes, introduction the notion of cost the suggested Actions, and filtering the Action Sets based on cost, to further reduce the computational time, and provide the most interesting and usable Action Rules.

REFERENCES

[1] S. Ramachandran A.A. Tzacheva, C.C. Sankar and 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)*. IEEE Computer Society.

[2] Rakesh Agrawal, Ramakrishnan Srikant, et al. 1994. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, Vol. 1215. 487–499.

[3] Mamoun Al-Mardini, Ayman Hajja, Lina Clover, David Olaleye, Youngjin Park, Jay Paulson, and Yang Xiao. 2016. Reduction of Hospital Readmissions through Clustering Based Actionable Knowledge Mining. In *Web Intelligence (WI), 2016 IEEE/WIC/ACM International Conference on*. IEEE, 444–448.

[4] A. Bagavathi, P. Mummoju, K. Tarnowska, A. A. Tzacheva, and Z. W. Ras. 2017. SARGS method for distributed actionable pattern mining using spark. In *2017 IEEE International Conference on Big Data (Big Data)*. 4272–4281. https://doi.org/10.1109/BigData.2017.8258454

[5] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.

[6] Michael Hahsler and Radoslaw Karpienko. 2017. Visualizing association rules in hierarchical groups. *Journal of Business Economics* 87, 3 (2017), 317–335.

[7] Seunghyun Im and Zbigniew W Raś. 2008. Action rule extraction from a decision table: ARED. In *International Symposium on Methodologies for Intelligent Systems*. Springer, 160–168.

[8] S.R. Marepally J.W. Grzymala-Busse and 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, 261–276.

[9] Jieyan Kuang, Albert Daniel, Jill Johnston, and Zbigniew W Raś. 2014. Hierarchically structured recommender system for improving NPS of a company. In *International Conference on Rough Sets and Current Trends in Computing*. Springer, 347–357.

[10] M. Lichman. 2013. *UCI Machine Learning Repository*. Technical Report. Irvine, CA, USA.

[11] Ming-Yen Lin, Pei-Yu Lee, and Sue-Chen Hsueh. 2012. Apriori-based frequent itemset mining algorithms on MapReduce. In *Proceedings of the 6th international conference on ubiquitous information management and communication*. ACM, 76.

[12] Maria Malek and Hubert Kadima. 2013. Searching frequent itemsets by clustering data: Towards a parallel approach using mapreduce. In *Web Information Systems Engineering–WISE 2011 and 2012 Workshops*. Springer, 251–258.

[13] Hongjian Qiu, Rong Gu, Chunfeng Yuan, and Yihua Huang. 2014. Yafim: a parallel frequent itemset mining algorithm with spark. In *Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*. IEEE, 1664–1671.

[14] Zbigniew W Ras, Agnieszka Dardzinska, Li-Shiang Tsay, and Hanna Wasyluk. 2008. Association action rules. In *Data Mining Workshops, 2008. ICDMW'08. IEEE International Conference on*. IEEE, 283–290.

[15] Zbigniew W Ras and Alicja Wieczorkowska. 2000. Action-Rules: How to increase profit of a company. In *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, 587–592.

[16] Zbigniew W Raś, Elżbieta Wyrzykowska, and Hanna Wasyluk. 2007. ARAS: Action rules discovery based on agglomerative strategy. In *International Workshop on Mining Complex Data*. Springer, 196–208.

[17] Sanjay Rathee, Manohar Kaul, and Arti Kashyap. 2015. R-Apriori: an efficient apriori based algorithm on spark. In *Proceedings of the 8th Workshop on Ph. D. Workshop in Information and Knowledge Management*. ACM, 27–34.

[18] Matteo Riondato, Justin A DeBrabant, Rodrigo Fonseca, and Eli Upfal. 2012. PARMA: a parallel randomized algorithm for approximate association rules mining in MapReduce. In *Proceedings of the 21st ACM international conference on Information and knowledge management*. ACM, 85–94.

[19] Li-Shiang Tsay* and Zbigniew W Raś. 2005. Action rules discovery: system DEAR2, method and experiments. *Journal of Experimental & Theoretical Artificial Intelligence* 17, 1-2 (2005), 119–128.

[20] Angelina A Tzacheva and Zbigniew W Ras. 2010. Association action rules and action paths triggered by meta-actions. In *Granular Computing (GrC), 2010 IEEE International Conference on*. IEEE, 772–776.

[21] Le Wang, Lin Feng, Jing Zhang, and Pengyu Liao. 2014. An efficient algorithm of frequent itemsets mining based on mapreduce. *JOURNAL OF INFORMATION &COMPUTATIONAL SCIENCE* 11, 8 (2014), 2809–2816.

[22] Xian Wu, Wei Fan, Jing Peng, Kun Zhang, and Yong Yu. 2015. Iterative sampling based frequent itemset mining for big data. *International Journal of Machine Learning and Cybernetics* 6, 6 (2015), 875–882.

[23] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI'12)*. USENIX Association, Berkeley, CA, USA, 2–2. http://dl.acm.org/citation.cfm?id=2228298.2228301