Action Rules for Sentiment Analysis on Twitter Data using Spark

Jaishree Ranganathan, Allen S. Irudayaraj, Angelina A. Tzacheva
Department of Computer Science
University of North Carolina at Charlotte
Charlotte, NC
{jrangan1, airudaya, aatzache}@uncc.edu

***Abstract*—Action Rules are vital data mining method for gaining actionable knowledge from the datasets. Meta actions are the sub-actions to the Action Rules, which intends to change the attribute value of an object, under consideration, to attain the desirable value. The essence of this paper to propose a new optimized and more promising system, in terms of speed and efficiency, for generating meta-actions by implementing Specific Action Rule discovery based on Grabbing strategy (SARGS) algorithm. For this, we perform a comparative analysis of meta-actions generating algorithmic implementation in Apache Spark driven system and conventional Hadoop driven system using the Twitter social networking data and evaluate the results. We perform corpus based Sentimental Analysis of social networking data, and test the total time taken by both the systems and their sub components for the data processing. Results show faster computational time for Spark system compared to Hadoop MapReduce for implementing the meta-action generation methods**.

Keywords— Sentiment Analysis, Natural Language Processing, Action Rules, Meta-Actions, Apache Spark, Hadoop MapReduce

I. INTRODUCTION

Data mining techniques are used to analyze huge data sets, to identify the underlying data patterns and to reveal the hidden knowledge. Data digitization in social networking and the extensibility of the platform for social networking, from micro devices like watches and smart phones to macro devices like desktops and laptops, have greatly contributed to the huge amount of structured and unstructured data that can be processed to generate sensible and meaningful information. Action-ability extends the concept of data analysis to a level further, by which the user can attain his/her intended action through deducing the Action Rules from the dataset.

The attributes in a dataset are divided into flexible attributes, whose value is mutable, and stable attributes, whose values is immutable [1]. The Action Rules are specific data patterns extracted from huge dataset which intends to change the current value of the flexible attribute, under consideration, to a desired value. An association rule is a rule extracted from an information system that describes a cascading effect of changes of attribute values listed in the left-hand side of a rule [6] on changes of attribute values listed in its right-hand side.

New algorithms have been proposed in the past decade to find some special actions based on the discovered patterns in the form of Action Rules. Action Rules propose an actionable knowledge that the user can undertake to his/her advantage. An Action Rule extracted from a decision system describes a possible transition of an object from one state to another state with respect to the distinguished attribute called decision attribute [11]. Action Rules have established its applications in variety of industries like healthcare, automotive, advertising etc. Some businesses require Action Rules generation on batch data and some require the same on streaming data. Hence cost of a time is the critical parameter to be considered for the algorithms which are proposed for generating these Action Rules. Authors in [3] [5] [11] [12] [13] [14] proposed variety of algorithms to extract Action Rules from the given dataset. The eccentric exponential increase in the data in recent years, causes delay in computations on tasks that are dependent on Action Rules and thus causing applications relied on Action Rules to be slow. Hence, this mandate need to develop viable, scalable, time efficient and distributed methods to work on such huge volume of data for generating action.

Distributed database systems are the most appropriate system to handle huge data sets. They have substantiated the reliability and efficiency for storage and processing bulk data sets. Apache provides various open source like Hadoop [4], Spark [21] [8], Hive, and Pig to process and handle huge data in the distributed system [2]. Hadoop is a distributed computing framework, to work with large datasets, across multiple computers, using a single programming model in a parallel fashion. This parallel processing aspect of the distributed computing plays a vital role in the cost of the processing time. Hadoop aims to provide scalable and fault tolerant computations on the given data. The main components of Hadoop are HDFS [17], YARN [20] and MapReduce [4].

Hadoop Distributed File System - HDFS is the data storage unit of the MapReduce operations. HDFS also keeps track of machines holding the data for a job [17]. Yet another Resource Negotiator - YARN [20] is an extra feature to the upgraded version Apache Hadoop framework. YARN supports multiple applications like MapReduce, Spark [8] [21], Storm, etc.

MapReduce is an open source cluster computing framework which uses HDFS to save and process huge data sets. The MapReduce framework works in such a way that it

2375-9259/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDMW.2017.14

divides the input data into size mutable input splits and cascades them to the clusters [Hadoop performance prediction]. By default, the input splits are 64MB individually. The MapReduce works in 2 phases, map and reduce. In the map phase, the input splits are processed in parallel fashion in the cluster and the intermediate results are stored in the cluster. In the reduce phase the intermediate results are combined and saved in the HDFS. The frequent access to the HDFS system makes it less suitable for iterative algorithms, which might require more map and reduce cycles.

Apache Spark addresses the issue with the concept of Resilient Distributed Dataset. Its in-memory data operations makes it well-suited for applications involving iterative machine learning and graph algorithms. Thus, we move our algorithm to Spark framework on top of Hadoop Distributed File System (HDFS) cluster. In this paper, we present a system SARGS (Specific Action Rule discovery based on Grabbing Strategy) which is an alternative to ARoGS [13] and implement the system in Spark like our old system MR-Random Forest algorithm for Distributed Action rule Discovery [18] using Hadoop MapReduce, either of them to extract Action Rules from the twitter data in the HDFS. The primary intent of the Action Rules generated is to provide viable suggestions to make a twitter user positive. Finally, we compare our current proposed system against our old Hadoop system of extracting Action Rules.

The rest of this paper is organized as follows. Section II gives the related works. In Section III, we discuss about the algorithms and technologies we used for this system. Section IV presents experiments and results in which give the current system results and compare it with the previous system. Section V concludes the paper.

II. Related Work

The social media data mining algorithms can be divided into two well established categories: Supervised Learning and Unsupervised Learning.

Supervised learning algorithms, provide prior knowledge of the class attributes for datasets. This supervised learning is further classified as classification and regression. When class attribute is discrete it is classification. Decision tree learning, naïve Bayes classifier, k-nearest neighbor classifier, classification with network information are classification methods. When class attribute is continuous it is regression. Linear regression and logistic regression are regression methods.

Unsupervised learning, the dataset has no class attribute and the task is to find similar instances and find significant patterns in dataset. For example, it can be used to identify events on Twitter data, because the frequency of tweeting is different for various events. Also by this method tweets can be grouped based on the times at which they appear and hence, identify the tweets' corresponding real world events. This section describes related research works in this area.

A. Sentiment Analysis and Twitter Data

The following research papers primarily performed sentimental analysis on twitter data.

Authors A. Balahur et.al [7] employs hybrid approach, using supervised learning with Support Vector Machines Sequential Minimal optimization (Platt 1998) linear kernel, on unigram and bigram features, but exploiting as features sentiment dictionaries, emotion lists, slang lists and other social media emotion features for a lexicon based sentimental analysis on the twitter data. The analysis involves two phases, preprocessing and then sentiment classifications. The processed tweets are then passed through the sentiment classification module. Training models were developed on the cluster of computers using Weka data software.

Authors A. Agarwal et.al [9] performed sentimental analysis on the twitter data. As part of the paper, they primarily experimented three types of models, unigram model, a feature based model and a tree kernel based model for two classification tasks, binary task which classifies the sentiment to positive and negative and 3-way task which classifies the sentiment to neutral along with the positive and negative category. The twitter data is first preprocessed using emotion dictionary, acronym dictionary and stop word dictionary. The comparative analysis on the models by experiment proved that tree kernel and feature based models outperform the unigram baseline.

Authors A. Chellal et.al [10] proposed multi-criterion real time tweet summarization based upon adaptive method. This method provides new relevant and non-redundant information about an event as soon as it occurs. The tweets selection is based on the following three criterions: informativeness, novelty and relevance with regards of the user's interest which are combined as conjunctive condition. Experiments were carried out on TREC MB RTF-2015 data set.

Authors Yu. Xu et.al [15] proposed methods to infer a user's expertise based on their posts on the popular micro-blogging site twitter. They proposed a sentiment-weighted and topic relation-regularized learning model. Sentiment intensity of a tweet is used to evaluate user's expertise and the relatedness between expertise topics is exploited to model inference problem. The following four common metrics were used for evaluation: accuracy, precision, recall and F1- score.

Authors F. Marquez et.al [24] proposed a simple model for transferring sentiment labels from words to tweets and vice versa by representing both tweets and words using feature vectors residing in the same feature space. Tweet centroid model developed in this paper outperformed the classification performance of the popular emoticon-based method for data labelling and better results than a classifier trained from tweets labelled based on the polarity of their words.

Authors M. Al-Ayyoub and I. Alsmadi [23] proposed a lexicon based sentiment analysis of Arabic tweets. This method is based on Sentiment Analysis and Opinion mining of social network data twitter feedbacks and comments. Unsupervised approach of sentiment analysis was applied which built a sentiment lexicon SA tool. This sentiment lexicon was built

with about 120,000 Arabic terms and a SA tool based on predicate calculus.

B. Action Rules Mining

The following research papers deal with Action Rule mining.

Action Rules was first introduced in [11] by Z. W. Ras and A. Wieczorkowska. Action Rules have been extracted using two approaches for more than a decade. One approach is using rule-based approach which extracts intermediate classification rules using algorithms like LERS [5] (to extract classification rules from complete information system) or ERID [3] (to extract rules from incomplete information system) from which Action Rules can be extracted using system DEAR [11] (uses two classification rules to get Action Rules) or system ARoGS [13] (uses single classification rule to extract Action Rules). Second approach is object-based -approach to extract Action Rules directly from the information system, without pre-existing classification rules, using system ARED [6] or Association Action Rules [14]. In this paper, we focus on rule-based approach much like ARoGS [13] and to generate Action Rules.

Performance prediction on Hadoop based distribution systems are generally carried out in two ways. First approach is the machine learning approach, which is often used to predict system performance leveraging past system execution data [9] and can achieve reasonable prediction accuracy. But this requires training the dataset. Second approach is the modelling based approach. Unlike the machine learning approach, modeling based approaches predict performance through modeling system behavior [9], and often can provide a better understanding regarding internal execution of a program and resulting performance.

C. Performance Prediction Analysis for Distributed Frameworks

The following research papers deal with performance prediction analysis for distributed processing frameworks.

Authors G. Song et.al [26] propose a framework to predict the performance of a Hadoop MapReduce job. The framework comprises of two modules which are, a lightweight job analyzer module and prediction module. The job analyzer module analyzes the submitted job and collects features related to the jobs and parameters related to the clusters. The prediction module makes use of the collected parameters to train the developed local linear model using local weighted linear regression method. The experimental results vouch the accuracy of the method's performance prediction.

Authors K. Wang et.al [25] propose a framework to predict the performance of Spark jobs. They apply analytical approaches to predict the performance of Apache Spark jobs. They leverage the multi-stage execution structure of Apache Spark jobs to develop hierarchical models that can effectively capture the execution behavior of different execution stages. They predict the job performance based on the limited scale execution job performance data on cluster. The experimental results show that the prediction accuracy evaluated for iterative and non- iterative algorithms is found to be high for execution time and memory, however the I/O cost prediction varied for different applications.

Authors A. Tzacheva and J. Ranganathan [1] have performed sentimental analysis on the twitter data via Action Rules generated. They implemented the ARoGS [14] algorithm proposed by Ras and Wyrzykowska, to generate the Action Rules. The real-time twitter data is extracted from the twitter API and then fed into the Hadoop Distributed File System – HDFS using MapReduce. Stanford core Natural Language Processing – NLP library was used to identify the polarity attribute through parts of speech (POS) of the tweets. MapReduce programs implementing the ARoGS algorithm was implemented on the twitter data. Experiments were conducted to generate Action Rules to make the user positive and increase the friends count.

In this work, we adapt the above described Action Rules mining algorithm [13] for twitter data processing. We follow model based approach to evaluate the existing MapReduce model [1] and our proposed Spark model for the Action Rule mining implementation on twitter data. We extend the work proposed by A. Tzacheva and J. Ranganathan [1] by incorporating Spark model for Action Rule mining and comparing performance with the MapReduce model. The proposed model involves simulation of the Action Rules generation by modifying the number of nodes in the cluster. Further, we perform sentiment analysis of twitter data based on discovering actionable patterns through Action Rules.

III. Methodology

The primary focus of this work is to evaluate the proposed Spark driven system implementing the Action Rule mining algorithm on twitter data, for making the users more positive, against the existing MapReduce driven system. The Action Rule mining comprises of the six phases: data collection, pre-processing, classification, sentiment analysis, Action Rule generation, Summarization.

A. Pre-Processing

In Pre-processing phase, we perform discretization on the following attributes, friends count and followers count by placing their values into intervals. As part of this phase, we perform data cleaning for missing values, feature selection and remove unnecessary values. We retained the following attributes Retweet Count, Is Favorited, User ID, Tweet Text, User Language, User Friends Count, User Favorites Count, and User Followers Count.

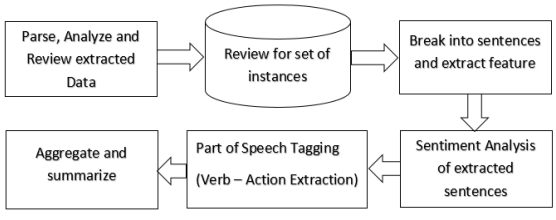

Fig. 1. Actionable Pattern Mining system for Twitter Sentiment Analysis

B. Feature Augmentation

In this phase, we add two additional attributes to the existing attribute set, first is sentiment attribute which can take the following values: positive, negative, neutral, very positive, very negative and the second is action attribute with attribute verb for actionable pattern mining because verbs suggest actionable knowledge. The latter was taken from the extracted part of speech from the tweets.

Stanford core Natural Language Processing – NLP library powered by Java was used for sentiment analysis. This NLP suite provides set of natural language analysis tools. The basic distribution provides model files for the analysis of well-edited English, but the engine is compatible with models for other languages. [16] This NLP suite provides annotators making use of java's Unicode support, by default UTF-8 encoding but also supports any character encoding. Out of these annotators we are using Tokenizer, part-of-speech, Sentiment Analysis in our work. We apply the POS - Part of speech annotator to label tokens with their part-of-speech (POS) tag, using a maximum entropy POS tagger.

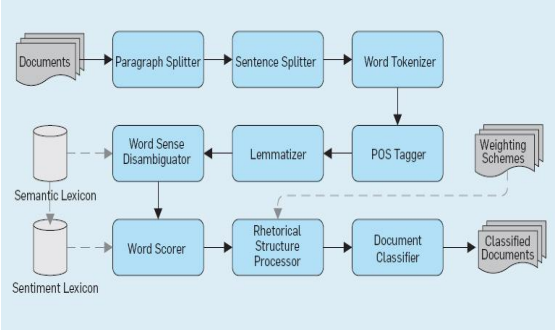

Fig. 2. Sentiment Analysis

Input: Grab your referral link today!

POS Tagger: Grab/VB your/PRP$ Referral/NN Link/NN today/NN !/.

Fig. 3. Part-of-SpeechTtagger - Verbs

C. Classification

We used LERS [27] algorithm to extract classification rules from twitter data. Classified each tweet as positive, negative, neutral, very positive, very negative. LERS [27] is a Learning from Examples based on Rough Sets which we use to extract classification rules from the information system. Our implementation follows distributed strategy of generating classification rules using LERS system. Fig 4. Gives the LERS algorithm. Using the information system S from Table I., 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 classification rules. LERS produces a set of certain and possible rules [27]. 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 during the following iterations.

For the information system given in Table 1, consider the following as decision support:

$(d1) * = \{x1, x2, x5, x8\}$ (1)

$(d2) * = \{x3, x4, x6, x7\}$ (2)

TABLE I. SAMPLE INFORMATION SYSTEM

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

ALGORITHM 1:
LERS (a_S, d_S)
(where, a_S and d_S are <key,value> pairs to store < distinct attribute values, objects in the information system supporting them >)
a_S store all attribute values other than decision values
d_S store all decision attribute values

fixedSupport ← a_S
while a_S is not empty **do**
  for each key, value pair in the a_S **do**
    if value is a subset of one of the values of d_S **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 a_S
  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 a_S
      end
    end
  end
end

Fig. 4. LERS Algorithm

LERS module given in Fig.4. For the given information system S, extracts certain and possible rules which are given in Table II.

TABLE II. LERS EXAMPLE FOR INFORMATION SYSTEM S

Iteration	Attribute Value Support	Certain rules	Possible Rules
1	$(a_1)^* = \{x_1, x_8\}$ - marked $(a_2)^* = \{x_3, x_4, x_5, x_6, x_7\}$ $(a_3)^* = \{x_2\}$ - marked $(b_1)^* = \{x_1, x_2, x_5, x_7\}$ $(b_2)^* = \{x_3, x_4, x_6, x_8\}$ $(c_1)^* = \{x_1, x_2, x_3, x_5, x_6\}$ $(c_2)^* = \{x_4, x_7, x_8\}$	$a_1 \rightarrow d_1$ $a_3 \rightarrow d_1$	$a_2 \rightarrow d_1$ $a_2 \rightarrow d_2$ $b_1 \rightarrow d_1$ $b_1 \rightarrow d_2$ $b_2 \rightarrow d_1$ $b_2 \rightarrow d_2$ $c_1 \rightarrow d_1$ $c_1 \rightarrow d_2$ $c_2 \rightarrow d_1$ $c_2 \rightarrow d_2$
	$(a_2, b_1)^* = \{x_5, x_7\}$ $(a_2, b_2)^* = \{x_3, x_4, x_6\}$ – marked $(a_2, c_1)^* = \{x_3, x_5, x_6\}$ $(a_2, c_2)^* = \{x_4, x_7\}$ - marked $(b_1, c_1)^* = \{x_1, x_2, x_5\}$ - marked $(b_1, c_2)^* = \{x_7\}$ - marked $(b_2, c_1)^* = \{x_3, x_6\}$ - marked $(b_2, c_2)^* = \{x_4, x_8\}$	$a_2 \wedge b_2 \rightarrow d_2$ $a_2 \wedge c_2 \rightarrow d_2$ $b_1 \wedge c_1 \rightarrow d_1$ $b_1 \wedge c_2 \rightarrow d_2$ $b_2 \wedge c_2 \rightarrow d_2$	$a_2 \wedge b_1 \rightarrow d_1$ $a_2 \wedge b_1 \rightarrow d_2$ $a_2 \wedge c_1 \rightarrow d_1$ $a_2 \wedge c_1 \rightarrow d_2$ $b_2 \wedge c_2 \rightarrow d_1$ $b_2 \wedge c_2 \rightarrow d_2$
	$(a_2, b_1, c_1)^* = \{x_5\}$ – marked	$a_2 \wedge b_1 \wedge c_1 \rightarrow d_1$	

D. Actionable Pattern Mining – Action Rules

ARoGS is Action Rules Discovery Based on Grabbing Strategy, which uses LERS. It was given by Ras and Wyrzykowska in paper [14] as an alternative to system DEAR [19] which extracts Action Rules from a pair of classification rules. The foremost advantage of using ARoGS is that it uses single classification rule to provoke Action Rules. ARoGS uses LERS kind of algorithm to extract Action Rules, without the need of verifying the validity of the certain relations. It just should check if these relations are marked previously by LERS. ARoGS presumes that system LERS construct classification rules describing target decision value. Fig 4. And Fig 5. Together gives the algorithm of ARoGS Fig. 6.

ALGORITHM 2:

```
AR (certainRules, decisionFrom, decisionTo)
  (where certainRules is provided by the LERS)
  for each key, value pair in the certainRules do
      if value1 equals decisionTo then
          actions ← empty list
              for each attribute value a in key do
                  A←attributeName(a)
                  actions. Add ("(A, → a)")
          end
      end
      Output actions as action rule
      ARoGS (actions, decisionFrom, decisionTo)
  end
```

Fig. 5. AR (Action Rules) Algorithm in distributed environment using MapReduce

Algorithm AR takes each candidate classification rule and form an Action Rule schema which in turn is given to the algorithm ARoGS to build a cluster of Action Rules around each schema. For the classification rules in Table II, algorithm AR generates following set of Action Rule schema:

AR1 (d1 → d2) =

$$(A, \rightarrow a2) \wedge (B, \rightarrow b2) \rightarrow (D, d1 \rightarrow d2) \quad (3)$$

AR2 (d1 → d2) =

$$(A, \rightarrow a2) \wedge (C, c2) \rightarrow (D, d1 \rightarrow d2) \quad (4)$$

AR3 (d1 → d2) =

$$(B, \rightarrow b1) \wedge (C, c2) \rightarrow (D, d1 \rightarrow d2) \quad (5)$$

AR4 (d1 → d2) =

$$(B, \rightarrow b2) \wedge (C, c1) \rightarrow (D, d1 \rightarrow d2) \quad (6)$$

Algorithm ARoGS Fig. 6 takes each Action Rule schema and using their flexible and stable attributes, generates following Action Rules which imply d1 → d2. For the Action Rule schema ARs1, the algorithm ARoGS finds all missing flexible attributes AM: {a1, a3, b1}. Each missing flexible attribute is filled into appropriate action terms. In ARoGS, the maximum number of Action Rules generated = AM. For ARs1, ARoGS produces following Action Rules:

AR1 (d1 → d2) =

$$(A, a1 \rightarrow a2) \wedge (B, \rightarrow b2) \rightarrow (D, d1 \rightarrow d2) \quad (7)$$

AR2 (d1 → d2) =

$$(A, a3 \rightarrow a2) \wedge (B, \rightarrow b2) \rightarrow (D, d1 \rightarrow d2) \quad (8)$$

Let an Action Rule R takes a form of:

$$(Y1 \rightarrow Y2) \rightarrow (Z1 \rightarrow Z2) \quad (9)$$

Where,
Y is the condition part of R
Z is the decision part of R
Y1 is a set of all left side of the all condition action terms
Y1 is a set of all right side of the all condition action terms
Z1 is the decision attribute value on left side
Z2 is the decision attribute value on right side

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 *stableValues* ∩ *decisionFrom*
missingValues ←set of missing flexible attribute values of the flexible attributes 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* ⊆ *actionsSupport*
then
Add *value* to *actions*
Output *actions* as action rule
end
end

Fig. 6. ARoGs (Action Rules Discovery Based on Grabbing Strategy) in a distributed environment using MapReduce

E. Support and Confidence of Action Rules

Let an Action Rule R takes a form of: (Y1 → Y2) → (Z1 → Z2) where, Y is the condition part of R, Z is the decision part of R.

Y1 is a set of all left side of the all condition action terms Y2 is a set of all right side of the all condition action terms Z1 is the decision attribute value on left side Z2 is the decision attribute value on right side.

In [13], the support and confidence of an Action Rule R is given as

$$Support(R) = \min \{card (Y1 \cap Z1), card (Y2 \cap Z2)\} \quad (10)$$

$$Confidence(R) = [card (Y1 \cap Z1)]/ Card (Y1) \cdot [card (Y2 \cap Z2)] / card (Y2) \quad (11)$$

In this paper, we use the following support and confidence formula given by Tzacheva et.al [18] to reduce the complexity.

$$Support(R) = \{card (Y2 \cap Z2) \quad (12)$$

$$Confidence(R) = [card (Y2 \cap Z2)] /Card (Y2) \quad (13)$$

F. Distributed Actionable Pattern Mining – MR Random Forest Hadoop

MR - Random-Forest algorithm for distributed Action Rules discovery using Apache Hadoop framework [22] and Google MapReduce [29]. An overview of the proposed algorithm is shown on Fig 8. We take as an input a set of files: the data, the attribute names, user specified parameters such as: minimum support, and confidence thresholds, 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).

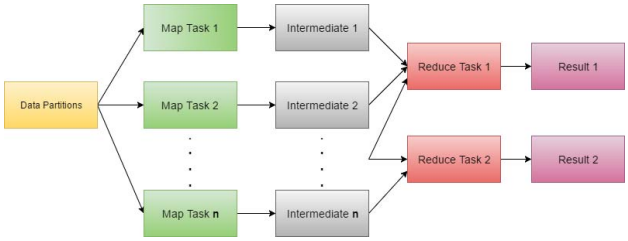

Fig. 7. Overview of MapReduce execution. The data partitions and results from Map and Reduce tasks reside in the distributed file system. The Map tasks and Reduce tasks are done in the distributed systems in a parallel fashion

G. SARGS (Specific Action Rule Discovery Based on Grabbing Strategy)

The Action Rules generated in section B comprises only one specific action terms. The left side of other action terms are empty. These Action Rules can give only a limited knowledge to the user and leave the clueless due to the lack of specific action terms. In this paper, we propose a new algorithm Specific Action Rule discovery based on Grabbing Strategy (SARGS) as an alternative to the algorithm ARoGS to fill all missing values in the Action Rule schema in an efficient time. Algorithm SARGS takes each Action Rule schema and finds all missing flexible attribute values. The algorithm then combines each missing value with other values giving a complete set of values that can fill all missing(left-side) of the conditional part of the Action Rule. For example, consider the Action Rule schema ARs1 shown in Section B. The algorithm SARGS finds all missing flexible attribute values AM: {{a1, a3}, {b1}}. Note that AM takes a form of main set containing a collection of multiple sets.

The algorithm then combines each element in the inner set with other elements in other inner sets. Thus, we get a combination of attribute values AC 1 = {a1, b1} and AC 2 = {a3, b1}. The algorithm puts each combination into

corresponding Action Rule schema to generate following Action Rules:

New AR (Action Rules) Algorithm in a distributed environment using MapReduce

AR3: (d1 → d2) =

(A, a1 → a2). (B, b1 → b2) → (D, d1 →d2) (14)

AR4: (d1 → d2) =

(A, a3 → a2). (B, b1 → b2) → (D, d1→d2) (15)

Unlike the algorithm ARoGS, algorithm SARGS does not produce any incomplete Action Rules. Instead it provides more specific Action Rules.

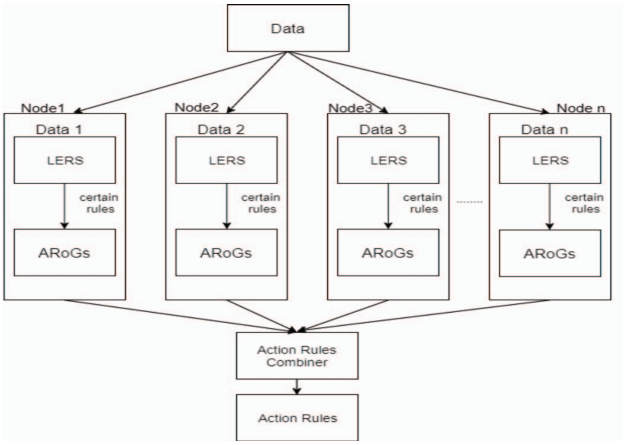

Fig. 8. MR – Random Forest Algorithm for Distributed Action Rules Discovery

H. Distributed Action Rule Mining in Spark

Spark [21] is a framework like MapReduce [4] to process large quantity of data in a short span of time. Spark introduces a distributed memory abstraction method called Resilient Distributed Datasets (RDD). Spark framework can outperform Hadoop MapReduce because of its in-memory capability, especially for iterative algorithms. Sparks performs as shown in Fig. 9. In [18], Hadoop manages data distribution over the nodes in a cluster and all algorithms ARoGS [13] and Association Action Rules [14], are implemented using MapReduce. When Hadoop manages data distribution, there are some possibilities that all records of single decision value move to a single Partition which can cause some loss of valuable Action Rules. In this paper, we propose a method similar to stratified sampling for data distribution to all partitions. We split the given data into groups where each group consists of records matching single decision value. We then measure how much proportion of data each decision value takes. According to this proportion, we take random samples of data from each group. By this way, each partition contains same proportion of data which is equal to the original dataset.

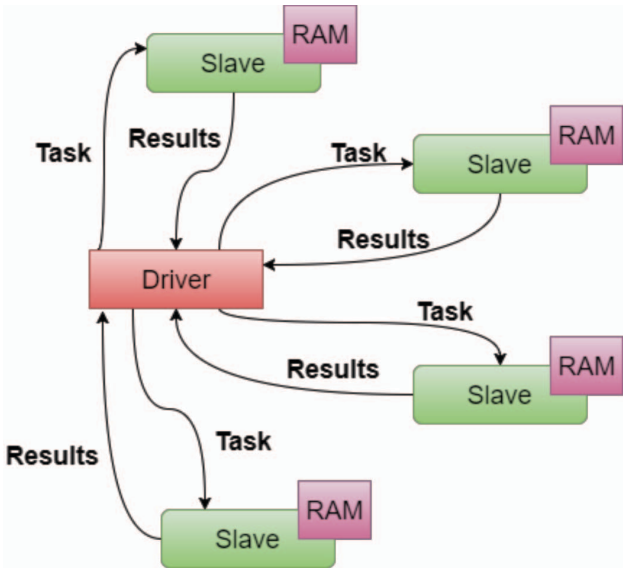

Fig. 9. Overview of Spark execution using Resilient Distributed Datasets (RDD). Tasks such as transformations are given to the slave nodes. Slaves after performing the tasks, cache the result in RAM. Results can be given back to the Driver node.

Fig 10. Shows an example data partition for the information system S shown in Table1. Our algorithms LERS and SARGS executes on each of these partitions and final Action Rules are grouped together. In Spark, reading each file: attributes, parameters and data creates three different RDDs. We manually split the data file into'd' files, where d is a distinct number of decision values. Each file contains samples of records from the given data file. Spark on reading each of these files create'd' RDDs. We also broadcast RDDs created from reading attributes and parameters file, so that all nodes can access them. Algorithms LERS and SARGS runs on each of d RDDs using Map Partition function, which is used to perform computations on each and every partition of data, and results in their own set of Action Rules with support and confidence. All Action Rules from the Map Partition function are sorted by the attribute name and returned as (Key, Value) pairs. We chose Action Rule to be a Key and support and confidence pair to be a Value. We then use groupByKey method to group all supports and confidences of a single Action Rule and aggregate them to calculate final support 'fs' and confidence 'fc' of an Action Rule. We output these Action Rules to a text file if fs > = minimumSupport and fc > = minimumConfidence. Now we describe the LERS, ARoGS algorithms and new SARGS method in detail. Consider an information system S:

S = (X, A, VA) (16)

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 represents a set of values for each attribute in A.

Example, VB =b0, b2.

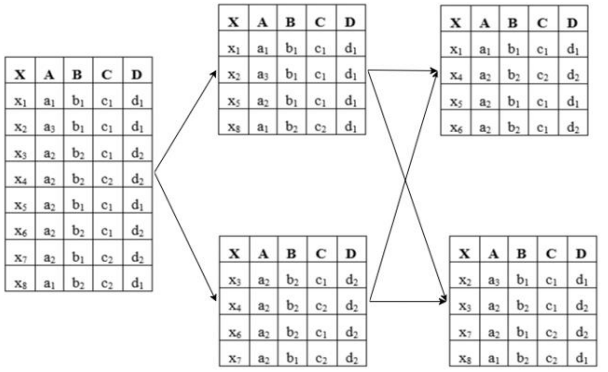

Fig. 10. Data Distribution to partitions

We use the sample information system S shown in Table I to demonstrate outputs from the above-mentioned algorithms. Consider attribute C to be a Stable Attribute, attributes A, B to be Flexible Attributes, attribute D to be the Decision Attribute, and that the user desires the decision value to change from d1 to d2. Also, consider that the user is interested in Action Rules with minimum support of 1 and minimum confidence of 80%. Instead of giving the data entirely to the Spark, we do some pre-processing step to make partitions of data to be given to Spark. All algorithms are then made to run on each partition of data. Following sub- sections talk about the Spark framework, algorithms LERS, ARoGS and SARGS and our implementation of these algorithms in Spark in a distributed environment.

IV. Experiments and Results

The Action Rules generated as part of the experiment focuses on suggesting how to improve emotions from negative to positive, neutral to positive and to increase the friends count. For this experiment, we used live tweets extracted using Twitter Search API on the latest tweets. The Twitter Search API searches against a sampling of recent tweets published in the past 7 days. Our data contains the following attributes: Retweet count, IsFavorited, User ID, Friends count, Favorites count, Followers count, Tweet text, User language, Tweet sentiment, Tweet verb. We analyzed 40,000 instances with 9 attributes. Table III and IV. gives the description about the dataset such as number of instances, attribute names, decision attribute values and data size. The Hadoop research cluster at University of North Carolina Charlotte was used to perform the experiments. This cluster has 6 nodes connected via 10 gigabits per second Ethernet network.

TABLE III. Property of Datasets

SNo	Property	Twitter Data	
1	# of instances	40,000	
2	Attributes	9	
3	Decision attribute values	**Tweet Sentiment**	**UserFriendsCount**
		Positive, Negative, Neutral	Increased numeric value than current

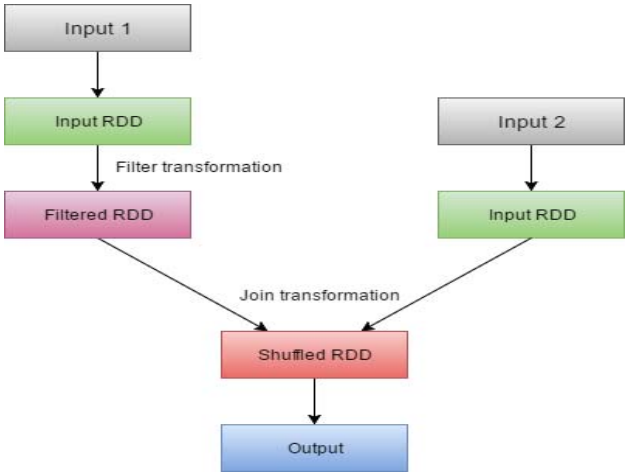

Fig. 11. Spark Lineage Graph Example

TABLE IV. Sample Data with Sentiment Analysis Results

Reweet	IsFavorited	UserId	FriendsCount
0	FALSE	898290540	283
0	FALSE	262194433	860

FavouritesCount	Followers Count	UserLanguage
242	62	En
302	688	En

Tweet Text	Tweet sentiment	Verb
RAY OF SUNSHINE	Neutral	NULL
LOVE OF MY LIFE	Neutral	NULL

We used Action Rules to suggest how to change from positive to negative and neutral to negative sentiment. Also, to change from lower number of friends count to higher number of friends. Three experiments were conducted on both Hadoop and the Spark systems, for improving the emotions of the users from neutral to positive, negative to positive and to improve the friends and followers count. The results are tabulated and the details of each experiments are debriefed below:

A. Experiment1

This experiment is focused in improving the user friends and followers count. The input attribute details are as follows: Stable Attributes are User Id and UserLanguage; Decision attribute is UserFriendsCount; Minimum support is 2 and confidence is 60%. The sample Action Rule generated for the experiment is recorded in the table V.

B. *Experiment2*

This experiment is focused in transforming the tweet sentiment attribute value from negative to positive. The input attribute details are as follows: Stable Attribute is UserLanguage; Decision attribute is Tweet Sentiment; Minimum support is 2 and confidence is 60%. The sample Action Rule generated for the experiment is recorded in the table VI.

C. *Experiment3*

This experiment is focused in transforming the tweet sentiment attribute value from neutral to positive. The input attribute details are as follows: Stable Attribute is UserLanguage; Decision attribute is Tweet Sentiment; Minimum support is 2 and confidence is 60%. The sample Action Rule generated for the experiment is recorded in the table VII.

TABLE V. SAMPLE ACTION RULE GENERATED BY THE SYSTEM FOR EXPERIMENT 1 CHANGE FROM CLASS USERFRIENDSCOUNT: LOW TO HIGHER NUMBER OF FRIENDS

MR Random on Hadoop (Action Rules)	SARGS on Spark (Action Rules)
TweetSentiment TweetSentiment Positive->TweetSentimentNeutral) ^(UserFollowersCount UserFollowersCount0-100- >UserFollowersCount1001- 5000)^(UserFavouritesCount UserFavouritesCount601-700- >UserFavouritesCount0- 100)^(IsFavourited -> IsFavouritedFALSE)= => UserFriendCount "UserFriendsCount0- 100->UserFriendsCount1001- 5000)[Support:- 3; NewConfidence:- 100%,OldConfidence:- 100%]"	(UserFavourites Count,100-200- >30000- Above)^(UserFoll owersCount, 0- 100->901-1000)^ (UserLanguage=fr) = => (UserFriends Count,0-100 -> 1001-5000) [Support:-3, Old-Confidence:-77%, New-confidence:- 100%

TABLE VI. SAMPLE ACTION RULE GENERATED BY THE SYSTEM FOR EXPERIMENT 2 CHANGE CLASS TWEET SENTIMENT FROM NEGATIVE TO POSITIVE

MR Random on Hadoop (Action Rules)	**SARGS on Spark (Action Rules)**
(UserFavoritesCount UserFavoritesCount10001-15000- >UserFavoritesCount101-200)^(UserFollowersCount UserFollowersCount0-100- >UserFollowersCount201-300) ==> (TweetSentiment "TweetSentimentNegative- >TweetSentimentPositive) [Support:- 2 ; New Confidence:- 100.0% ; Old Confidence:- 100.0%]"	(UserFavoritesCount, 501-600 -> 30000-Above) ^ (UserFollowersCount, 201-300 -> 5001-10000) ^ (UserFriendsCount, 301-400 -> 201-300) ^ (UserLanguage = en) => (TweetSentiment, Negative -> Positive) [Support: 2, Old Confidence: 100%, New Confidence: 100%]

TABLE VII. SAMPLE ACTION RULE GENERATED BY THE SYSTEM FOR EXPERIMENT 3 CHANGE CLASS TWEET SENTIMENT FROM NEUTRAL TO POSITIVE

MR Random on Hadoop (Action Rules)	**SARGS on Spark (Action Rules)**
(UserFavoritesCount UserFavoritesCount10001-15000- >UserFavoritesCount1001-5000)^(IsFavorited IsFavoritedFALSE- >IsFavoritedFALSE)^(UserFollowers Count UserFollowersCount1001-5000->UserFollowersCount301-400)^(UserFriendsCount UserFriendsCount301-400- >UserFriendsCount101-200) ==> (TweetSentiment"TweetSentimentNeutral->TweetSentimentPositive) [Support:- 2 ; New Confidence:- 100.0% ; Old Confidence:- 100.0%]"	(UserFavoritesCount, 401-500 -> 30000-Above) ^ (UserFollowersCount , 0-100 -> 5001-10000) ^ (UserFriendsCount, 101-200 -> 201-300) ^ (UserLanguage = en)=>(TweetSentiment, Neutral -> Positive) [Support: 2, Old Confidence: 69%, New Confidence: 100%]

The experimental results explaining the time taken for the system to generate the Action Rules and the number of Action Rules generated are tabulated in the table VIII.

TABLE VIII. DURATION AND ACTION RULES COUNT

Exp	**#nodes**	**Hadoop**		**Spark**	
		minutes	Action Rules count	minutes	Action Rules count
1	1			1.38	75
	4	1.25	213		
2	1			1.17	2345
	4	3.52	614		
3	1			1.75	1071
	4	4.47	357		

The Action Rules are assessed using the support and confidence metrics. User specified threshold of support 2, and confidence 60% were applied.

V. CONCLUSION

In this paper, we proposed a new Spark system implementing the upgraded algorithm Specific Action Rule discovery based on Grabbing Strategy (SARGS) as an optimized alternative to system ARoGS [14] to extract complete Action Rules like system DEAR [11], ARED [5] and Association Action Rules [6]. The reduced time cost for our system in comparison with the conventional Hadoop system for distributed Action Rule mining attributes to the Apache Spark's ability to perform in-memory computations and reduced communication cost compared to Hadoop MapReduce. We have also given more appropriate way of partitioning the data to be given to multiple nodes to extract Action Rules from them.

In future, we plan to introduce more robust and automated method of data sampling based not only on the decision

attribute but also on stable and flexible attributes also. We also plan to test our system with more real-time large data like NPS dataset to test and improve system's scalability and feasibility. Also, we plan to introduce the notion of cost of the Action Rules generated from the distributed environment.

REFERENCES

[1] A.A. Tzacheva and J. Ranganathan, "Action Rules for sentimental analysis using Twitter", International Journal of Social Network Mining, 2017, in press.

[2] A. Bagavathi, A.A. Tzacheva, "Rule based Systems in Distributed Environment: Survey", in Proceedings of International Conference on Cloud Computing and Applications (CCA17), 3rd World Congress on Electrical Engineering and Computer Systems and Science (EECSS'17),June 4-6 2017, Rome, Italy, pp 1-17

[3] A. Dardzinska, Z.W. Ras, "Extracting Rules from Incomplete Decision Systems: System ERID", in Foundations and Novel Approaches in Data Mining, (Eds. T.Y. Lin, S. Ohsuga, C.J. Liau, X. Hu), Advances in Soft Computing, Vol. 9, Springer,2006, 143-154

[4] 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.

[5] 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.

[6] Z.W. Ra´s, A. Dardzi´nska, 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.

[7] A. Balahur, "Sentimental Analysis in social media texts" European Commission Joint Research Centre Vie E. Fermi 2749 21027 Ispra (VA), Italy

[8] M. Zaharia et al. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. NSDI 2012, pp. 15-28.

[9] Apoorv Agarwal, Boyi Xie, Ilia Vovsha, Owen Rambow and Rebecca Passonneau, "Sentiment Analysis of Twitter Data" Workshop on Language in Social Media LSM, Portland, Oregon, USA, 2011, pp. 30-38.

[10] A. Chellal, M. Boughanem and B. Dousset, "Multi-criterion real time tweet summarization based upon adapive threshold," 2016 IEEE/WIC/ACM International Conference on Web Inteligence, pp. 264-271

[11] 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.

[12] 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.

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

[14] 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.

[15] Y. Xu, D. Zhou and S. Lawless, "Inferring your expertise from twitter: Integrating Sentiment and Topic Relatedness," 2016 IEEE/WIC/ACM International Conference on Web Intelligence, pp 121-128

[16] M. Christopher D., M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pp. 55-60.

[17] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. 2010. The Hadoop Distributed File System. In Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Tech- nologies (MSST) (MSST '10). IEEE Computer Society, Washington, DC, USA, pp. 1-10.

[18] A.A. Tzacheva., A. Bagavathi, and P.D. Ganesan, "MR - Random Forest Algorithm for Distributed Action Rules Discovery", in International Journal of Data Mining & Knowledge Management Process (IJDKP), 2016, Vol. 6, No. 5., pp.15-30.

[19] 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.

[20] V. K. Vavilapalli, A. C. Murthy, C. Douglas, et. al. 2013. Apache Hadoop YARN: yet another resource negotiator. In Proceedings of the 4th annual Symposium on Cloud Computing (SOCC '13). ACM, New York, NY, USA, Article 5, 16 pages

[21] M. Zaharia et al. "Spark: Cluster Computing with Working Sets", HotCloud 2010.

[22] 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, pp 113-120

[23] M. Al-Ayyoub and I. Alsmadi, "Lexicon-based sentiment analysis of Arabic tweets," Int. J. Social Network Mining, Vol. 2, No.2, 2015, pp 101-114

[24] F. Bravo-Marquez, E. Frank and B. Pfahringer, "From opinion lexicons to sentiment classification of tweets and vice versa: a transfer learning approach," 2016 IEEE/WIC/ACM International Conference on Web Intelligence, pp 145-152

[25] K. Wang and M. Maifi Hasan Khan, "Performance prediction for Apache Spark platform" 2015 IEEE 17th International Conference on High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), and 2015 IEEE 12th International Conf on Embedded Software and Systems (ICESS), pp 166-173

[26] G. Song, Z. Meng, F. Huet, F. Magoule`s, L. Yu and X. Lin, "A Hadoop MapReduce Performance Prediction Method" 2013 IEEE International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing, pp-820-825

[27] 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.

[28] 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.