Proceedings of the 3rd World Congress on Electrical Engineering and Computer Systems and Science (EECSS'17)
Rome, Italy – June 4 – 6, 2017
Paper No. CCA 107
ISSN: 2369-811X
DOI: 10.11159/cca17.107

Rule Based Systems in a Distributed Environment: Survey

Arunkumar Bagavathi, Angelina A. Tzacheva
Department of Computer Science, University of North Carolina at Charlotte
9201 University City Blvd, Charlotte, NC, 28223, USA
abagavat@uncc.edu; aatzache@uncc.edu

Abstract **-** Over the past years, the internet has become faster, computer storage has become larger and the data from internet users and sensors is piling up to larger amount and is also spread around the globe. This requires more time and space for a single computer to cope with them. Ecosystems like Hadoop helps to store, process and retrieve back the data efficiently in a distributed fashion. Data mining finds substantial improvements over such distributed frameworks to process a large volume of data in a lesser time. Currently, there are many approaches to do data mining tasks such as classification and clustering in a distributed setup using Hadoop MapReduce, Spark, and other Cloud platforms. Actionable pattern mining is a rule based data mining approach for discovering knowledge from information systems in a form of Action Rules. An emphasis of traditional classification rules from a supervised Machine Learning is to predict class label of a data object. Whereas Action Rules produce actionable knowledge in the form of suggestions on how an object can change from one class value to another more desirable class value. This paper gives a brief survey of previous works on association and classification rule mining algorithms in a distributed environment, as well as action rule mining algorithms, and discusses Action Rule Mining in a distributed environment.

Keywords**:** Hadoop, MapReduce, Spark, Action Rules, Classification Rules, Distributed Action Rule mining

1. Introduction

The towering production of data in the recent years, due to increased usage of web, social media and IoT, has led to the age of big data. Hence, there is a demand for ideal data mining, machine learning and business intelligence tasks to retrieve valid knowledge from such a massive data. Also, because of rise in the usage of cloud storage and cloud based services, data is distributed over multiple computers. Most of the data mining algorithms are computationally intensive specifically for the big data and needs more resources to engage huge data.

Apache has provided many open source frameworks like Hadoop, Spark, Hive, etc. to process and manage such bulk data [1]. The merits of these frameworks are that anyone can form a cluster with their own set of computers to perform parallel computations on a cluster. These frameworks distribute the data in the cluster and also split the work given to them to multiple nodes or computers in a cluster, each of which runs on their own part of the data. Finally, when all nodes finish executing their tasks, all results merge to give final set of results. Many computational intensive fields like Machine learning, Data mining, Image processing, Bioinformatics, Recommendation systems, Spam detection benefit from the parallel processing frameworks because of their capability to do distributed and parallel computations in a fault tolerant manner.

Knowledge discovery is a process of preprocessing and transforming a given data such that previously unidentified knowledge can result after mining the transformed data. Some of the data mining algorithms are association – to find frequently associated data in a dataset, classification – a supervised machine learning algorithm to classify the available data into one or more classes and clustering – an unsupervised machine learning algorithm to group similar data into a cluster. These algorithms produce tremendous amount of information from the big data regardless of whether they are of user's interest and most of such knowledge goes undetected. In this paper, the focus is mainly on rule based machine learning algorithms like classification rules and their implementation in parallel processing frameworks.

Action Rules on the other hand, extracted based on user's interest from a dataset describes a possible transition of data from one state to another, or in other words, Action Rules reclassify data from one category to another. Action Rules can help many applications like in online shopping sites, Action Rules help suggesting how to improve customer satisfaction for a product or in medical field, Action Rules can help suggesting how to improve a patient's health. In [2] [3] [4] [5] [6]

authors propose a variety of algorithms to extract Action Rules from a given dataset. Since data is growing at a rapid pace, the proposed action rule mining approaches are stumbling to produce Action Rules in an expected time limit. This causes prolonged execution of the tasks that are dependent on results of action rule mining algorithms and producing results. This paper reviews some of the methods proposed previously to extract knowledge with classical rule based algorithms such as classification rules [7] [8] [10] [13] and association rules [9] [11] [12] in cloud computing frameworks, using MapReduce [14] and Spark [15], as well as commercial Cloud platforms such as Microsoft Azure [16], Amazon Web Services (AWS) [17], and GoogleCloud [18]. This paper also reviews some of the Action Rule extraction algorithms [2] [6] [5]. Finally, we discuss the possibility to extract Action Rules using selected parallel processing frameworks.

2. Distributed Computing Frameworks

Two most common and famous parallel processing frameworks are Apache Hadoop [19] MapReduce [14] and Apache Spark [15]. As mentioned in [19], "Hadoop is a framework that allows distributed processing of large datasets across clusters of computers using single programming model". MapReduce and Spark are the programming models to process large datasets in a parallel fashion on top of Hadoop. One of the important components in Hadoop is Hadoop Distributed File System (HDFS) [20], which splits the large data and manage small data chunks in multiple nodes. MapReduce and Spark can work on top of HDFS to access and work with those small portions of data. Since the tasks run on such small chunks of data and all tasks run in parallel, total computation time reduces comparatively to the same task running on the large dataset in a single machine. Features other than distributed and parallel computations of these frameworks such as their architectures, data management are beyond the scope of this paper. In the following sections, we discuss parallel processing design of these selected frameworks.

2.1. Hadoop MapReduce

J. Dean and S. Ghemawat, from Google, in [14] proposed a parallel computing programming model called MapReduce, which has a potential to process large datasets in parallel with distributed algorithms in a fault tolerance way in a cluster of nodes or computers. The authors claim that the goal of MapReduce is to make users to think about how to do put their methods or algorithms into MapReduce programming model instead of considering other complex functionalities such as data distribution, task parallelization, load balancing and fault-tolerance. Hadoop run these complex tasks in background of a given MapReduce program. Hadoop MapReduce always works on top of Hadoop Distributed File System (HDFS), a distributed file system to store the data as multiple splits in different locations.

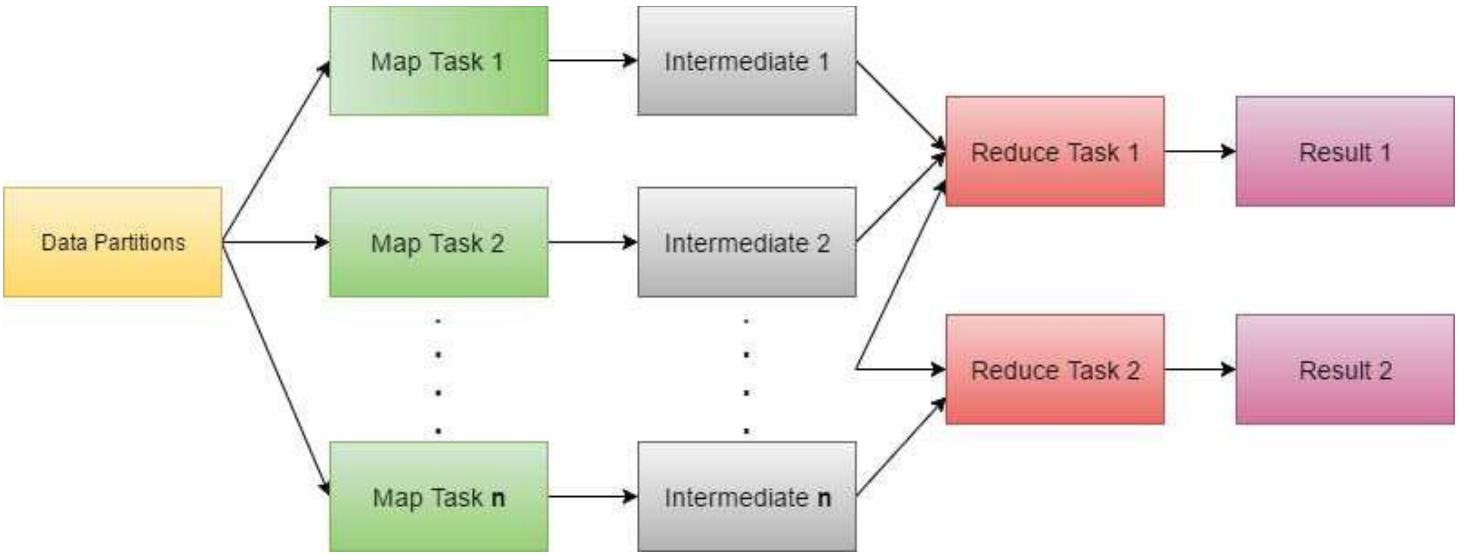

Fig. 1: Overview of MapReduce execution. The data partitions and results from Map and Reduce tasks reside in the HDFS. The Map tasks and Reduce tasks in the distributed systems run in a parallel fashion.

MapReduce works with two functions: Map and Reduce. Both functions take input and produces output as <Key, Value> pairs. A Master node monitors the whole MapReduce execution process. Master node finds appropriate number of

available nodes currently and assign them as slave nodes. Hadoop [19] preserves data locality by assuring that distance between data node and slave node is minimum. Initially, the master node allocates slave nodes for running map tasks. Now the slave nodes getting their own tasks, MapReduce starts its Map phase on them. Since the map function runs on a small split of a large data, the results are usually intermediate outputs. Each map function writes their intermediate output <Key, Value> pairs back into HDFS. The master node again finds an available slave node(s) to perform reduce task. Reduce function collects all values for a single key from the map task, sorts them, perform computations like aggregation on them and writes the output as <Key, Value> pair to the distributed file system. Other MapReduce tasks can use this result or other applications can take this result to perform other computations. Fig.1 gives an overview of the execution of a MapReduce phase.

MapReduce runs in a master slave architecture such that master keeps track of all slaves both in HDFS and in MapReduce tasks. Slaves in turn keeps sending heartbeat signal to the master to notify that it is still working. During any system or node failure, slave node stops sending the signal and the master can allocate the failed task to some other slave node. In the HDFS, each data chunks exists in three different locations. Therefore, even if one node that is having the data is down, Hadoop can get the data from other locations.

2.2. Spark

Even though MapReduce form a good framework to deal with distributed parallel operations and it has some fault tolerance scheme, it has some disadvantages of writing and reading operations of intermediate outputs on the Hadoop Distributed File System. This causes frequent disk access for accessing and storing the data. This makes many iterative machine-learning algorithms like classification, clustering and regression to work inefficiently. Spark [15] introduce a distributed memory abstraction method called Resilient Distributed Datasets (RDD) to avoid frequent disk access from the data nodes. The intention of RDD is to do in-memory computations unlike Hadoop MapReduce [14] to store the results in disks for a huge dataset in a fault tolerant approach. Spark reads the data from the given source, split them into chunks and store them in node memory as an RDD. The driver node allocates tasks to the worker nodes where the data resides. A task can be either *transformation* or *action*. During *transformation* stage, Spark perform computations on a data split and the results are stored in-memory of other worker nodes. The result of all worker nodes together forms another RDD. *Action* stage collects the resulting RDDs and send the collection to the driver node or save the collection to a storage system like databases. Fig.2 gives an overview of the execution of Spark.

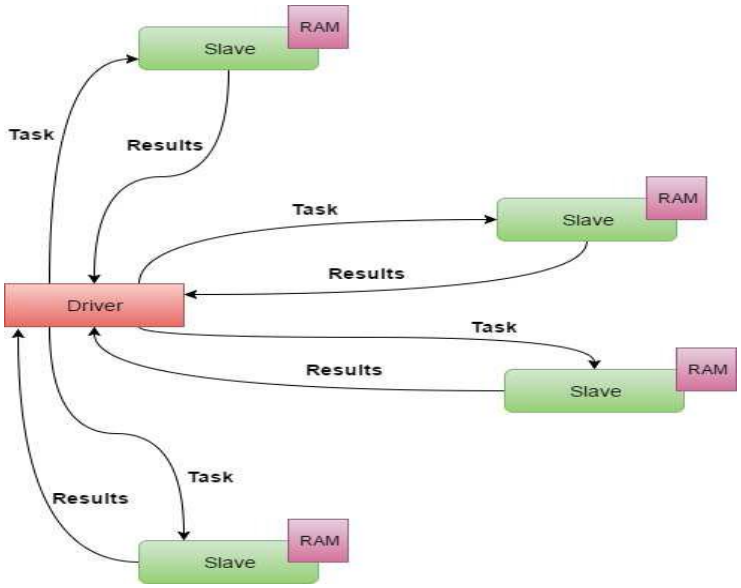

Fig. 2: Overview of Spark execution using Resilient Distributed Datasets (RDD). Slave nodes accept tasks such as transformations. After performing the tasks, slave nodes cache the result in RAM as an RDD. Other transformation tasks can use this RDD or the Driver node retrieve the results by initiating an action task.

Since the data and results are present in a system's memory, Spark can access such data much faster without any delay like in Hadoop MapReduce. Therefore, Spark is suitable for many iterative algorithms like machine learning algorithms. Spark handles fault tolerance by having a lineage graph of RDDs. Fig.3 shows a simple lineage graph of combining values from two inputs. When a node fails or certain portion of data is lost at a certain stage, Spark replays the failed or lost partition from the lineage graph and produce results to the lower levels. Because of the existence of lineage graph, Spark avoids data replication over multiple nodes like Hadoop MapReduce, which frees some space in the cluster that the RDDs can use.

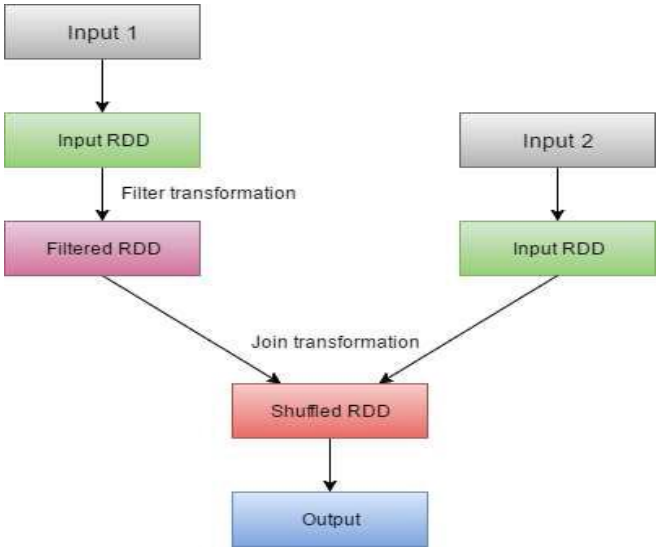

Fig. 3: Lineage graph of joining inputs from Input 1 with Input2. After reading from Input 1, Spark filters the read values to get only required values. Values from both inputs are combined and stored as an output.

Spark also provide many packages like Spark SQL, Spark Streaming, GraphX and MLlib [21]. Spark SQL can query any tables from databases like Hive, Cassandra, etc. In the other hand, it can also create tables in the databases from the raw data. Spark Streaming can manage a data stream from Kafka [22] or Twitter Stream. Spark collects the streaming data for small amount of time and create RDDs from the collected data that can be processed further using Spark SQL or MLlib. GraphX is the primary ground for graph processing and graph analytics. Fig.4 compares Hadoop MapReduce and Spark when running Logistic Regression and K-Means clustering algorithms in different number of machines for a 100GB of data [15].

2.3. Pig, Hive and HBase

Apache provides several database oriented frameworks like Pig [23], Hive [24] and HBase [25]. Apache Pig consist of its own data flow language: PigLatin. It provides an environment for data processing. Pig is similar to a scripting engine. Pig be used to connect SQL queries with other programming models such as MapReduce. Pig can process both structured and unstructured data. Hive is a data warehousing model useful for data summarization and analysis. It is used to load structured data into the Hadoop Distribute File System (HDFS), and provides SQL like queries called Hive Query Language (HiveQL) to query the data. Hive converts the queries into MapReduce jobs in the background, to fetch and process data from multiple nodes and return results. Apache HBase is a column oriented, non-relational distributed database to store the user data. HBase has no SQL language, and does not perform any data processing on its tables. It is useful for storing matrices of WebPage links, or document term frequency tables.

2.4. Flink, Storm and Kafka

Apart from Hadoop [14] and Spark [15], Apache provides specialized frameworks for big data stream processing, such as: Flink [26], Storm [27], Kafka [22] , SAMOA [28] and several others [1]. Previously mentioned frameworks like MapReduce and Spark are suitable particularly for batch processing systems, that is they work on already collected data but are not directly suitable for live streaming data. Apache Storm [27], is a tool specifically designed for streaming data

processing. Storm guarantees that all tuples from the data stream will be processed. Apache Flink [26] is an open source platform for both stream and batch processing, giving developers two-in-one benefit. Kafka [22] is a distributed data-streaming platform that acts as a broker to route the data reliably from the data producers to data consumers. Kafka processes the streaming data from the source, partitions them into topics, stores them in a log structure, from which multiple consumers can subscribe to a topic and read them at a same time. Kafka API can be easily used along with Spark, Storm or Flink.

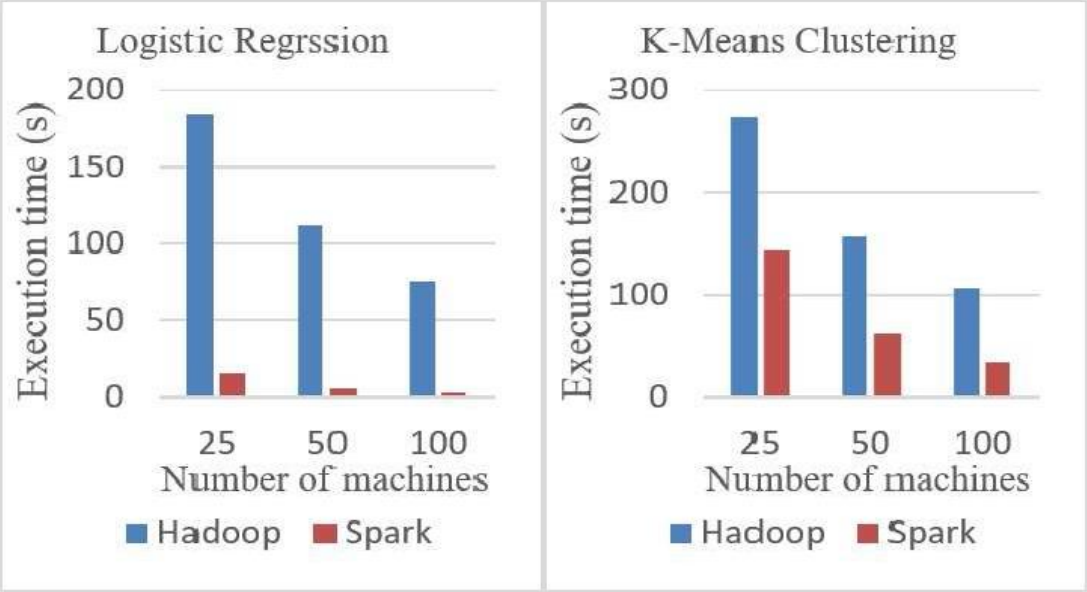

Fig. 4: Execution times of Logistic Regression and K-Means Clustering for 100GB of data.

3. Distributed Machine Learning

Most of machine learning algorithms are computationally expensive for tuning up its parameters and produce a final model that fits best for a particular dataset [29]. These algorithms, when taking large scale of data as an input, obtain large number of high dimensionality data instances, which increase complexities of the algorithms. Such algorithms require data and task parallelization to run efficiently on a huge volume of data. This section gives a review on some of the available frameworks for supporting distributed and parallel computation on machine learning. The distributed frameworks discussed in the previous section contain their own built in machine-learning library.

3.1. Spark MLlib

Apache Spark [15] has Spark MLlib [21] to create machine learning models in the distributed environment. MLlib is the distributed machine-learning library that provides simple and rich ecosystem for running many machine-learning algorithms including decision trees and forests, linear SVM, Naïve Bayes, linear regression, logistic regression, k-means clustering, Principle Component Analysis, and stochastic gradient descent. Spark, due to its in-memory computations feature, makes iterative algorithms to execute faster. Since many machine learning algorithms make series of iterations over a data, Spark is most suitable for many machine learning algorithms. Fig.4 Compares speed test of Hadoop and Spark for Logistic Regression and K-means clustering.

3.2. FlinkML

Similarly, Apache Flink [26] has FlinkML, a machine-learning library, which is still an Apache's incubating project provide data scientists a platform to create a model on a subset of local data and use the same model in a cluster for the streaming data measured in size megabytes or gigabytes or beyond. Inspired by Spark MLlib, FlinkML also provide facilities for many machine-learning algorithms.

3.3. Mahout

Apache provides a scalable framework Apache Mahout [30] dedicated for distributed machine learning on a batch of data. A notable feature in Mahout is the recommender engine, whose sole purpose is for recommendation and it also have other classification and clustering algorithms. Initially Mahout supported only Hadoop MapReduce [14] jobs. However, today it can bind with Apache Spark or Flink.

3.3. SAMOA

SAMOA [28] is another Apache incubator project, which is similar to Mahout, however it is a distributed stream processing machine-learning platform. It provides a pluggable machine learning programming abstractions for many data mining and machine learning tasks. It includes: classification, regression and clustering. It provides an option of implementing an algorithm and plugging it into multiple distributed stream processing engines such as: Spark, Flink, or Storm without changing a single line of code.

3.4. GraphLab

Apart from these frameworks, there are other frameworks dedicated for running distributed and parallel machine learning algorithms. One of such work is Distributed GraphLab [31], which performs parallel computations in both shared and distribute memory settings in a directed graph structure. It can be used for representing Neural Networks, for Page Ranking, Social Network mining, or any data in a graph format. Since machine learning models comprise dependencies in data, some parameters need to be updated iteratively for further computations. GraphLab allows users to assign data and computation for each vertex or machine and edge in the graph. Each vertex can interact with neighboring vertices. Since the communication is based on a graph structure, GraphLab can perform multiple MapReduce computations concurrently. The distributed GraphLab also provide option for storing global variables that is common for all the vertices. Their experimental results show that the system outperforms Hadoop by 20-60 times. In comparison with Spark [32], GraphLab outperforms Spark, which is more optimal than Hadoop due to in-memory computations, only in graph algorithms like PageRank and BFS. However, GraphLab provides almost equal performance to non-graph algorithms like SVM and K-means. In the field of graph processing, Google provides an open source framework: TensorFlow [33], to efficiently handle graphs for highly scalable deep learning problems.

3.5. Parameter Server

Parameter server [34] is another fault-tolerant framework for efficient feature extraction from the large scale of data distributed over multiple machines. Like MapReduce [14], Parameter server follows master-slave architecture. But, Parameter server maintains a server group and slave group, where each group can contain multiple machines. Each machine in the server group can communicate each other while each slave can communicate only with the respective master node. Master nodes distribute the data and tasks to the slave nodes. The results from the slave tasks are only local optimal. To get global optimality, each master node stores global parameters of an algorithm and replicate them to other master nodes for reliability and scaling, which the slave nodes can frequently access and update. This framework performs well with algorithms like regression, topic modelling and deep learning particularly when the data contains large number of parameters.

4. Commercial Cloud Services

In order to use all frameworks for machine learning and data mining algorithms in a distributed fashion, it is necessary to have an environment setup as a cluster of computers. The performance of algorithms improves if configuration of these computers is high. Setting up such a good configuration cluster costs more money and the memory of each computer is easily occupied if there are lot of jobs and the data is huge. Current technological trend has set a computation model called Cloud Computing to make general resources like CPU and storage available for leasing for a cost [35]. Cloud Service Providers offer software services (SaaS) or a platform to develop and host applications (PaaS) or an entire computer infrastructure like virtual machines (IaaS), which an end-user can utilize for a cost. Fig.5. shows several types of services that a service provider can provide, which forms the Cloud Computing architecture. SaaS are software services like gmail, facebook and YouTube, which are outside of the scope of this paper. Several large companies like Amazon, Google, IBM and Microsoft provide public Cloud Services, which anyone can use.

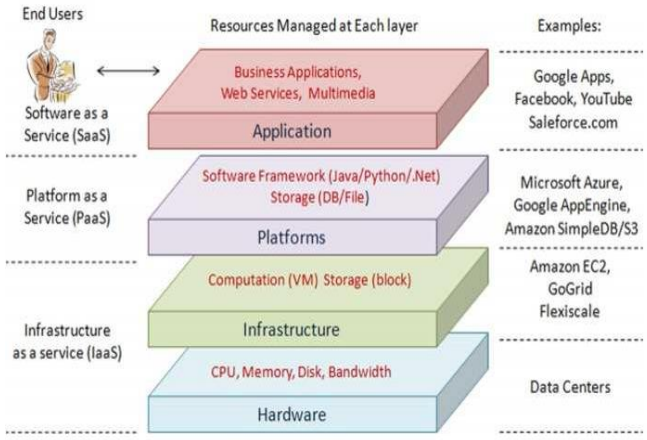

Fig. 5: Cloud Architecture.

4.1. Amazon Web Services

AWS provide many tools and services for developing and deploying user programs. For distributed computing, AWS has an option of Amazon Elastic MapReduce (EMR) [17]. EMR provides a Hadoop framework to process huge volume of data distributed across Amazon EC2 instances (virtual machines). They can accommodate other frameworks like Apache Spark and Flink. AWS also provide facilities for distributed data storage using Amazon Elastic File System to help data file management in an EMR cluster. AWS provide Amazon Machine Learning to build predictive models and deploy them.

4.2. Microsoft Azure

Microsoft Azure provides multiple products for machine learning, batch and stream processing jobs, cluster management, data storage, databases, IoT computing, etc. For cluster services, Microsoft Azure offers HDInsight [16], a cloud Hadoop that provide analytic clusters for Apache Spark, Flink, Storm, Kafka and Storm under Service-Level Agreement (SLA). It also provides high productivity platforms like Visual Studio, Eclipse for developers to code. To help data scientists to combine code, derive equations, and visualizations from their code, Azure provides popular notebooks like Jupyter and Zepplin. Azure also provides many APIs like Face Recognition, Anomaly detection, etc.

4.3. Google Cloud Platform

Google Cloud [18] provides cluster based, big data analytics and machine learning services. It also provides infrastructure called Google Compute Engine that allows users to launch Virtual Machines on demand. Google App Engine is a platform-as-a-service that allows the user to focus on their code instead of other operational details like infrastructure management. Like Microsoft Azure, Google Cloud ML (Machine Learning) platform provides APIs like Cloud NLP, Cloud Speech API, Cloud Vision API, etc. for machine learning purposes. Google Cloud ML also provides pre-trained machine learning models and eases to create our own models. Google Cloud ML provides neural-nets, which gives high training accuracy and performance comparatively with other platforms.

4.4. IBM Bluemix

IBM Bluemix [36] is another Cloud Computing platform, which provides many big data analytics tools and services. For cluster management, Bluemix provides BigInsights for setting up Apache Hadoop clusters within few minutes. IBM Watson is the most popular cloud service, which provides many useful APIs like Language Translator, Tone Analyzer, Speech to Text, NLP classifier, etc., for developing cognitive applications. Bluemix also has APIs for developing data analytics applications using Apache Hadoop and Spark. IBM Watson Machine Learning is an incubating project to create and deploy machine learning models. IBM Bluemix currently provides tools to develop programs using Python, Scala and R.

5. Distributed Rule Mining

Rule based Machine Learning intends to produce methods that identifies, learns or evolves 'rules' to store, manipulate knowledge. This differs from other machine learning algorithms, which identify a common model that can be utilized in the future to make predictions. In data mining, the more useful classification methods are the rule - based algorithms because of its simplest nature to understand and are easy to extract from the data records. Many algorithms are available to generate rules and help classifying the data records. Rules take the representation of:

IF conditions THEN result

The condition part of the rule act as an antecedent and the result part act as consequent. Due to racing volume of big data, many researches in the history have adopted distributed processing frameworks such as Hadoop MapReduce [14] and Spark [15] and help classification algorithms generate classification [7] [8] [10] [13] or association rules [9] [11] [12] in a brief time for a large volume of data.

5.1. Distributed Classification Rule Mining

Classification is a supervised machine learning process, in which the input data D is in the form of (X_i, Y_i), $i=1,2,...,N$, where X is a set of 'n' attributes $A_1, A_2, A_3, \ldots, A_n$ and each attribute A_k have their own values and Y is a class attribute which contains class values for each record in D. Classification algorithms read such input data and classify each object in the dataset to a certain class value. Classification rules is one of the classification methods that provide knowledge in a form of rules. Classification rules can be extracted using direct and indirect methods [10]. Direct method induces classification rules directly from the given dataset. Whereas, in indirect methods, the algorithms produce intermediate results like decision trees from the data from which in turn we can extract classification rules by tracking a path from the root of the decision tree to a leaf of the decision tree.

G. Wu et. al [13] proposed a distributed computation of combination of decision tree algorithm C4.5 [37] and ensemble learning method called *Bagging* [38] using MapReduce framework. C4.5 calculates *Entropy* and *Information Gain* for each attribute and choose splitting attribute As with high Information Gain as a root node r. After choosing the splitting attribute, the algorithm creates n branches giving n different nodes, where n is a number of distinct values in A_s. C4.5 algorithm [37] finds Entropy and Information Gain for the resulting branch or intermediate nodes and attribute splitting part continues for the branch nodes unless the node contains only one data record in it or if a node contains single class label instances. From the decision tree, we can form classification rules by tracking the paths from root node to all leaf nodes, that is we get one classification rule for each path in the tree. Thus, C4.5 is an indirect method of providing classification rules from decision trees. Let the classifier built from C4.5 algorithm for data D be φ(D). *Bagging* [38] is used to improve the accuracy of machine learning algorithms particularly for classification and regression. Bagging splits the data D into m sequence datasets $D_1, D_2, \ldots, D_m$ and the algorithm fills each dataset D_i using bootstrap sampling with replacement from D. Now the algorithm builds classifier φ on each D_i. When a test sample enters the system, classifiers $\varphi(D_i)$ takes x and the final class label of x is given using voting procedure from the results of all $\varphi(D_i)$. Bagging also helps avoiding overfitting problems. In [13], the algorithm split the data D into 'm' partitions where 'm' is number of mappers in Hadoop system. Algorithm C4.5 is used to build a base classifier C_i on each D_i partitions where $1 \leq i \leq m$. In the Reduce phase, bagging procedure collects all C_i classifiers and gives the test dataset to all the classifier to predict class label of records in the test dataset.

W. Dai and W. Ji [12] proposed a MapReduce implementation of traditional decision tree algorithm C4.5 [7]. Considering the communication cost in the MapReduce model, authors in [7] have created three data structures to store basic information such as:

- attribute table store attributes and their values and corresponding row_id and class value of the attribute values
- count table stores number of instances of each class label for each attribute value
- hash table to store link between the tree nodes

The attribute and count tables are filled while reading data from the dataset using single MapReduce phase. Remaining phases goes iteratively to compute information gain ratio of each attribute in all nodes. The attribute with maximum gain ratio is a splitting attribute and the algorithm updates hash table and count table during the end of each iteration. In this way, the algorithm builds a decision tree using a pipeline of MapReduce phases.

V. Kolias et. al [8] proposed a direct method of extracting classification rules in MapReduce framework [8]. This system employs two-step: first step reads the training examples and create a set of conditions that covers most of the training examples and second step combine the conditions from step 1 to form a rule and evaluate the rule. The best rule that covers maximum number of training examples is considered to the candidate rule. All training examples that is covered by the best rule is removed from the training examples before searching for next best rule. This helps to find the classification rules that covers maximum number of examples in limited iterations. Algorithm terminates once sufficient number of training examples are covered. In MapReduce model, the system uses a driver module that orchestrates 2 jobs, one for each step, to extract classification rules. This driver module is necessary for providing basic information of the dataset like attribute names, values and their types and it stores previous rules to check for the duplicate rules. Table 1 give evaluations on multiple datasets showing that the system RuleMR acquires better or equal accuracy comparatively to other classifiers like J48, OneR and Random Forest algorithms.

Table 1: Evaluation of RuleMR with other algorithms.

Dataset	RuleMR	OneR	J48	ID3	Rand. Forest
Breast Cancer	94.7	72.7	75.8	N/A	97.2
Car Evaluation	100	70.7	96.2	100	99.8
Weather	100	71.4	100	100	100
Mushroom	100	98.5	100	N/A	100
Vote	99.5	95.6	96.3	N/A	N/A

Even though only few methods have been proposed to extract classification rules using distributed frameworks, no evaluation on these methods were given. Since data is distributed into many partitions and each partition build their own classifier, evaluation of results comparing with the non-distributed system is required apart from acquiring better coverage and more accurate rules [8] and faster results.

5.2. Distributed Association Rule Mining

Association rules are similar to the classification rules discussed in the previous section but association rules notify relations among attributes in the datasets, which can be used in market basket problems. For example, to find patterns in customer transactions from a supermarket. In most recent years, association rules are also being used for classifying objects in a dataset. Since the data size is increasing rapidly in this era of big data, finding all possible relations among the attributes consume a lot of time. This requires a need of distributed and parallel approaches to find such patterns for iterative procedures such as Apriori algorithm [39]. Apriori algorithm is a bottom-up procedure to build frequent itemsets from the given dataset. Apriori algorithm reads the data and produces all 1-itemsets list. Then it goes through two steps: 1. Candidate itemset generation and 2. Pruning step. Step 1 generates itemsets of length *k* using the itemsets of length *(k-1)* from k^{th} iteration using join operation on each itemset. Step 1 thus produces candidate itemsets C_k. Step 2 removes all itemsets $c \in C_k$ such that *c* does not form a subset of at least one itemset from $(k-1)^{th}$ iteration. Apriori algorithm continues until it completes *n* iterations, where *n* is the number of attributes in the dataset or it stops when it does not produce any candidate itemsets. There have been few works on extracting association rules using Apriori algorithm in distributed environments like MapReduce and Spark, which are discussed below:

X. Lin [9] proposed a MapReduce [14] based approach to extract association rules [9]. This system works by splitting the data records horizontally into m partitions. The m mappers access their data partition and results in the format of <itemset, 1>. 'm' combiners collect data from their own mapper results and add the count value of a single attribute value to result in <itemset, count>. MapReduce then shuffles the obtained results into 'r' partitions, one to each reducer. Reducers sum up the count of an attribute value from all mappers to produce final result of <itemset, count>. Note that the length of the resulting itemset is equal to the iteration count. If the iteration count is 1, length of the itemset is also 1; if the iteration count is 2, then the length of the itemset will be 2. The resulting count is validated across the given minimum support. Only itemsets which matches the minimum support moves into next iteration (k+1). Iteration (k+1) use all

frequent itemsets from iteration k to perform candidate itemset generation. Pruning step uses the input data given to the operating mapper node and deletes some itemsets. The pruned itemsets advance to the reducer phase and the iteration goes on. This model operates in a single job for both candidate itemset generation and pruning, which increases communication cost.

Qiu et. al [11] modified the above discussed Apriori model and proposed Yet Another Frequent Itemset mining (YAFIM) in Spark framework [15]. This system operates in 2 phases. The first phase reads a transaction dataset, extract all frequent itemsets of length 1 from it and create an RDD. It also broadcasts the transaction database to all nodes. The next phase operates iteratively for (n-1) iterations using the input itemsets and broadcasted dataset to produce next set of frequent itemsets. This system uses the Spark's advantage of retaining the data in the memory to store the original dataset until the algorithm completing its process reducing communication and computation cost rapidly. Due to improved features in Spark and minor changes in the algorithm, this system outruns the MapReduce implementation of Apriori in [9].

Rathee et. al [12] proposed Reduced-Apriori or R-Apriori to speed up the algorithms proposed in [9] and [11] using Spark [15]. In this paper, the authors focus on the second phase of classical Apriori algorithm, which generates more candidate itemsets from singleton frequent itemsets, which is stored in hash tree to prune faster in the future. This step increases the time complexity for massive datasets. Reduced-Apriori removes the time-consuming candidate itemset generation and uses bloom filter instead of hash tree. Bloom filter is used to test whether a set contains a particular element. Bloom filter stores all itemsets from previous iteration. Each transaction in the dataset is made intersection with all itemsets in the filter such that the result of intersection contains only items, which exist in the filter. The algorithm then yields all possible pairs of itemsets in the pruned transaction. Addition of results from various nodes gives the final count of the new frequent itemset pair. Fig.6 shows the speed comparison of methods proposed in [9] [11] and [12] during all iterations. From these systems, it is notable that Spark, due to its capability of performing in-memory computations suits better for iterative data mining algorithms like Apriori.

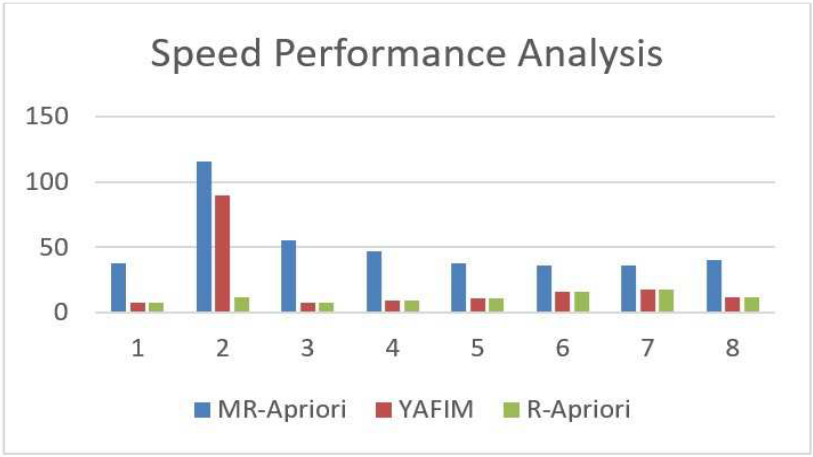

Fig. 6: Speed Performance Analysis in iterations of system MR-Apriori, YAFIM and R-Apriori.

5.3. Inductive Logic Programming

There are few other rule induction techniques like Inductive Logic Programming (ILP) in MapReduce [40]. ILP generate hypothesis by combining the background knowledge with positive and negative examples from the data and it tries to make relations among the provided data examples with the background domain knowledge. In other words, ILP is an automated method for extracting logic rules from data examples and background knowledge. Currently ILP does not play a crucial role in Action Rules mining algorithms [3] [4] [5], which are the attribute based algorithms. ILP may be used in the future to augment Action Rules extraction methods.

6. Action Rule Mining

Action Rules are desirable actionable patterns discovered from large amounts of data. They are preferable when a user would like to take action based on the discovered knowledge. Action Rules, like classification rules, has

antecedent, which are series of actions, and consequent, which is a decision action. The **antecedent** and **consequent** parts can give a hint to the user that he needs to perform certain actions on attributes of their data to get a desired result. More than a decade there has been a lot of research on diverse methods on generating action rules. So far, action rule mining is based on two approaches: rule-based approach [5] [6] and object-based approach [2] [3] [4].

6.1. Background

Information systems can form decision tables [5]. Information system is a set of objects and attributes where each attribute contains their own set of values. In case of decision tables, the attributes can be either condition attributes or a decision attribute and the condition attributes can be either stable or flexible attributes. Thus, decision table in terms of an information system can take a representation of,

$$S = \{U, A_{st}, A_{fl}, d\} \text{ where,} \tag{1}$$

U is a set of objects in the information system
A_{st} is a set of stable conditions/attributes
A_{fl} is a set of flexible attributes
d is a decision attribute where $d \notin \{A_{st} \cup A_{fl}\}$

Flexible attributes can change their values into another value of the same attribute whereas stable attributes remain constant once assigned. All action rule mining algorithms can extract actions rules only if the information system contains at least one flexible attribute. Action means a state in which a flexible attribute 'f' change its value from 'f_1' to 'f_2'. Thus, an action of attribute 'f' takes a form of $(f_1 \rightarrow f_2)$, where f_1 belongs to precondition set and f_2 belongs to postcondition set. Series of such actions of flexible attributes together with constant stable attributes constituting a resulting decision action form an action rule.

Table 2: Information System.

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

Consider Table 2 as a sample information system S throughout this section to give examples for each methodology. Consider attribute b as a stable attribute and attribute d as a decision attribute. For example, an action rule (r_1}) from S can take a form of:

$$r_1: \quad (a, a_0 \rightarrow a_1) \wedge (b, b_0) \rightarrow (d, d_2 \rightarrow d_1) \tag{2}$$

Support *Sup(r)* and Confidence *Conf(r)* of an action rule can be calculated using the following formula [4]:

$$Sup(r) = min\{card(Y_1 \cap Z_1), card(Y_2 \cap Z_2)\} \tag{3}$$

$$Conf(r) = [card(Y_1 \cap Z_1)/card(Y_1)] * [card(Y_2 \cap Z_2)/card(Y_2)] \tag{4}$$

where $Y \subseteq \{A_{st} \cup A_{fl}\}$ and $Z = d$. Y_1 is a set all left-hand side values of the conditional part and Y_2 is a set all right-hand side values of the conditional part. Z_1 is a left-hand side value of the decision action and Z_2 is a right-hand side value of the decision action. Support and Confidence of an action rule determines how valid is the action rule with the dataset for a particular decision action. For rule

$$r_1: Y_1 = \{a_0, b_0\},\ Y_2 = \{a_1, b_0\},\ Z_1 = d_2 \text{ and } Z_2 = d_1 \quad (5)$$

6.2. Rule-Based Approach

Rule-based approach of extracting Action Rules necessitates two steps: (1) finding patterns from the dataset in the form of classification rules and (2) generating Action Rules from the classification rules. There are many rule-based approaches to extract Action Rules from both complete and incomplete information systems. We discuss few of those works below:

All rule-based approaches in action rule mining use Learning from Examples using Rough Sets (LERS)[41] type of algorithm to extract classification rules. Unlike classification models like C4.5, which extracts classification rules from the intermediate decision trees, LERS is a direct method of extracting classification rules from complete decision tables without any intermediate results. LERS follows a bottom-up strategy to build up rules. All individual attribute values including decision attribute values and their corresponding objects are collected. Let A be a set of all attributes in a decision table, V_a be a set of values for $a \ \varepsilon \ A$ and X_v be the objects supporting an attribute value v. Thus, for the decision table in *Table 2*, $A = \{a, b, c, d\}$, $V_a = \{a_0, a_1, a_2\}$, $V_b = \{b_0, b_2\}$,…., $V_d = \{d_1, d_2\}$ and $X_{a0} = \{x_2, x_4\}$, $X_{a1} = \{x_1, x_5\}$,……, $X_{d1} = \{x_1, x_3, x_5\}$, $X_{d2} = \{x_2, x_4\}$. Objects supporting condition attributes X_c are marked and said to be *certain rules* if and only if $X_c \subseteq X_d$ where X_d is a set of objects supporting the decision attribute d. Remaining attributes are marked as *possible rules*. LERS algorithm again combine these possible rules to form a next set of attribute values of length 2. The algorithm follows previous procedures to get next set of *certain* and *possible rules*. When there are no *possible rules*, the LERS algorithm ends and lists certain rules as a list classification rules for the decision table. For the decision table provided in *Table 2*, the classification rules from LERS would be:

$$a_0 \rightarrow d_2\ ,\ a_1 \rightarrow d_1\ ,\ a_2 \rightarrow d_1\ ,\ b_2 \rightarrow d_1\ ,\ c_1 \rightarrow d_2 \quad (6)$$

Authors Tsay, et.al. [6] define an algorithm called Discovering E-Action Rules from Incomplete Information Systems (DEAR3), the third installment of system DEAR, which uses tree based approach to extract Action Rules from an incomplete information system. Incomplete information system means that the decision table contains some null values. DEAR 3 proposes a novel method – Classification rules discovery for an Incomplete Decision system (CID) to extract classification rules from an incomplete information system. CID first fills all missing values in decision table using a roulette wheel method. Roulette wheel consists of 'm' sections where m is the number of distinct values for an attribute for which there are missing values. The area of each section depends on the frequency of each value occurring in the decision table. For each missing value, the roulette wheel is rotated 'm' times and the CID algorithm choose a value that occurs more than (m/2) times on the top of the wheel for a single missing value. CID follows a method similar LERS [41] type of algorithm to extract classification rules from the complete decision table. In addition to finding certain and possible rules, CID calculates support of each rule. Next iteration in the algorithm takes only rules with support greater than or equal to the minimum support. DEAR3 uses two classification rules to extract single action rule.

Ras, et. al [5] gives an approach to produce Action Rules from a single classification rule with Action Rule Discovery based on Agglomerative Strategy (ARAS). This system works on an assumption that the provided information system is complete without any missing or null values. This system uses LERS [41] algorithm to generate classification rules. Consider the classification rules for the decision table in Table 2 generated by LERS and consider that the user prefers to change the decision from d1 to d2. From the available classification rules, ARAS first generates action rule schema. Action rule schema defines a pattern for an action rule from the classification rule. Since flexible attributes form a base for forming actions, the algorithm avoids classification rules without the flexible attributes for constructing Action Rules. For the certain rules from Table2, ARAS generates only one action rule schema ARs:

$$(\mathbf{a},\ \rightarrow a_0) \rightarrow (\mathbf{d},\ d_1 \rightarrow d_2) \quad (7)$$

For the action rule schema AR_s, let V_{st} be the stable attributes, V_{fl} be the flexible attributes, *decisionFrom* be the left side of the decision action (d_1 for the above action rule schema) in AR_s and let X_{ARs} be the objects in the decision table supporting $V_{st} \cup$ *decisionFrom.* Now, the ARAS algorithm takes all missing flexible and stable attribute values from the decision table and fill into the action rule schema to form a set of Action Rules *AR*. Let X_{AR} be the objects supporting *AR*.

Action rule *AR* is not given to the user if $X_{AR} \nsubseteq X_{ARs}$. Some of the Action Rules from system ARAS and DEAR 3 for the decision table *Table 2* are:

$$(\mathbf{a}, a_1 \rightarrow a_0) \rightarrow (\mathbf{d}, d_1 \rightarrow d_2)\ ;\ (\mathbf{a}, a_2 \rightarrow a_0) \rightarrow (\mathbf{d}, d_1 \rightarrow d_2)\ ;\ (\mathbf{a}, a_1 \rightarrow a_0) \wedge (\mathbf{c}, c_2) \rightarrow (\mathbf{d}, d_1 \rightarrow d_2) \tag{8}$$

Thus, ARAS system [5] treats each classification rule with target decision value as a seed and pulls all other classification rules with non-target decision values near that seed to form a cluster and produce all possible Action Rules from the cluster. In this way, the authors claim that the proposed system works much faster than system DEAR due to reduced number of comparisons between classification rules for extracting Action Rules.

6.3. Object-Based Approach

Object-based approaches as proposed in [2], [3] and [4], extract Action Rules directly from the information system without additional classification rules extraction like in rule-based approaches. We discuss few of the object-based approaches for discovering Action Rules below:

Authors Ras et. al [4] propose a method of extracting new form of Action Rules from the given information system S using Apriori like algorithm in the name of Association Action Rules. Like Apriori algorithm, this system generates single item pair and its support as an initial itemset. While forming a single itemset, stable attribute values just form as items while the flexible attributes values form as item actions. For the information system S given in Table~\ref{tab:2}, Association Action Rules algorithm generates following itemset pairs:

$$(A, a_0 \rightarrow a_1), (A, a_0 \rightarrow a_2), (B, b_0), (B, b_2), (D, d_1 \rightarrow d_2) \tag{9}$$

Only the itemsets whose support matches the given minimum support are considered as frequent itemsets (*Pruning step*) and are taken to the next iteration to combine with other frequent itemsets to combine *k*-element frequent itemset into (*k+1*)-element itemset (*Merging step*). The iteration continues until the algorithm finds *m*-element itemsets where *m* is the number of attributes in the information system *S* or if no itemsets come out of Pruning step. Once the iterations complete, the algorithm takes each frequent itemset containing the decision action and produce Action Rules.

Authors A. Hajja et. al [3] propose a new dimension of action rule as *Object-driven Action Rules*. This system works on object-driven information system. Information system *S* changes to Object-driven information system S_o when some instances in the information system belong to an object 'm_i'. This way, the instances can group into '*m*' clusters where '*m*' is the number of objects in the information system. In addition, this system introduces the notion of temporal constraint into the information system. The authors used medical data for this system. Each patient acts as an object. Each patient's records are ordered by their visit to the hospital (i.e) means that the records of patient's y^{th} visit occurs immediately after $(y\text{-}1)^{th}$ visit. Object-driven action rule uses Association Action Rules [18] to extract Action Rules from this new information system.

Table 3: Sample Information System for Object-Driven Action Rules.

	Object ID	A	B	C	D
X_0	1	a_1	b_1	c_1	d_1
X_1	1	a_2	b_1	c_1	d_1
X_2	1	a_2	b_2	c_2	d_2
X_3	1	a_1	b_2	c_1	d_1
X_4	1	a_2	b_1	c_1	d_2
X_5	2	a_1	b_2	c_1	d_2
X_6	2	a_2	b_1	c_1	d_1

Table 3 shows a simple information system for the object-driven Action Rules algorithm. Action Rules extraction algorithm in this approach take instances of each object as input and produces Action Rules for all m objects and finally

aggregating similar patterns of Action Rules from m objects. Thus, with this approach, the authors give an object-independency assumption for extracting Action Rules for individual objects where each object can have their own features or characteristics, which they do not share with other objects. Authors also claim that the system extract more accurate Action Rules for real world cases.

The object-driven approach in [3] can cause over-fitting problems by individualizing each object particularly when there are limited number of instances for an object. For example, with an information system of n instances, the maximum support of each action rule can be $(n/2)^2$, that is when half of the instances satisfy precondition of the action rule and the other half instances satisfy postcondition of the action rule. When these n instances are divided into m subsystems where each subsystem contains p instances which satisfy the condition $p < m < n$. Thus, it is obvious that the maximum support $(p/2)^2$ of the system after division reduces when the value of m (number of subsystem) continues to become higher. To handle these problems in object-driven Action Rules approach, A. Hajja et. al (2014) in [2] proposed a new algorithm that uses a combination of object-driven action rule approach and classical action rule mining approach. In this approach, the authors generalize or cluster some objects, which contains similar features, after categorizing individual objects. For the medical dataset with 225 unique objects or patients, the authors cluster the patients, who react similarly for given treatments, into 40 different subsystems. Table 4 shows the effect of clustering in terms of number of Action Rules and their total support from [2].

Table 4: Result comparison of Object-driven and Hierarchical Object-driven approach.

Decision Shift	No. of Action Rules		Total Support	
	Object-driven	40 clusters	Object-driven	40 clusters
$2 \rightarrow 1$	14	91133	28	931985
$1.5 \rightarrow 1$	388	10054	776	66874
$1 \rightarrow 0.5$	96	59927	200	497465
$1 \rightarrow 0$	954	85769	1996	755361

6.4. LISp-Miner

Rauch and Šimůnek [42] introduce a software LISP-Miner to extract G-Action Rules using a GUHA procedure: Act4ft-Miner. GUHA is an exploratory data analysis tool. GUHA procedure takes analyzed data, find patterns in the data and tests it with the input data. Act4ft-Miner procedure in this software extracts Action Rules as an advanced version of association rules. Act4ft-Miner produces G-Action Rule R in the form of:

$$\varphi_{St} \wedge \Phi_{CHg} \approx^* \psi_{St} \wedge \Psi_{CHg} \tag{10}$$

where, φ_{St} – stable antecedent Boolean attribute;

Φ_{CHg} – expression of change in flexible antecedent attributes;

ψ_{St} – stable consequent Boolean attribute;

Ψ_{CHg} - expression of change in flexible consequent attributes;

$\approx^*$- Act4ft quantifier

Lisp-Miner also provide some visualizations in the form of confusion matrix and its corresponding histogram for each extracted action rule. Fig.7 shows a sample visualization of a confusion matrix and corresponding histogram for an action rule.

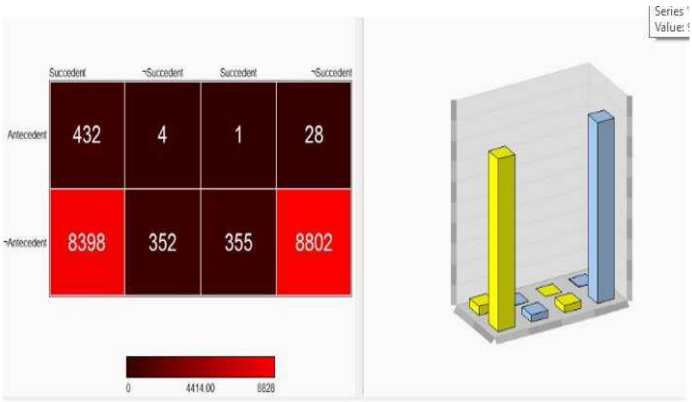

Fig. 7: Confusion Matrix and histogram of an action rule extracted from a BMI dataset.

7. Distributed Action Rule Mining

Nowadays, many data mining algorithms are starting to use parallel processing frameworks. Action Rules also need a method in order to be extracted efficiently with big data. Both rule-based and object-based Action rule extractions can make use of distributed computing frameworks [14] [15] [26] to generate Action Rules for large datasets quickly. Since the decision tables from which we get Action Rules consist of stable attributes and flexible attributes, random splitting of dataset may give imbalanced attribute values to the worker nodes. The algorithms that extract Action Rules from such partition of data result in low quality Action Rules. To extract good Action Rules from such distributed systems, it requires an intelligent way of sampling or distributing the data to worker nodes. Most popular sampling method is random sampling and its varieties like stratified sampling and cluster sampling. Each of the sampling technique selects the data randomly based on their own strategy. There has been active research on selecting subsets of the training data to give them to multiple working nodes using submodular functions [43] [44] [45] to intelligently split the data such that a model running on chunks of the split data is close to global optimal solution when the same model runs on a whole data. Consider a set N = {1,2,3,….,n}. Submodular function that is common in physics and mathematics is a set function that satisfies $f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$, $\forall A,B \subseteq N$, where f is a set function of the form $f: 2^N \rightarrow \mathbb{R}$. In submodular data partitioning method, data is partitioned into m partitions and a utility function f is assigned to each partition. The final goal is to have partitions of data that maximizes the overall utility. Accuracy of a Distributed Deep Neural networks (DNN) running for a phone classification dataset split into 30 partitions for both random sampling and submodular partitioning [45] is given in Figure 8.

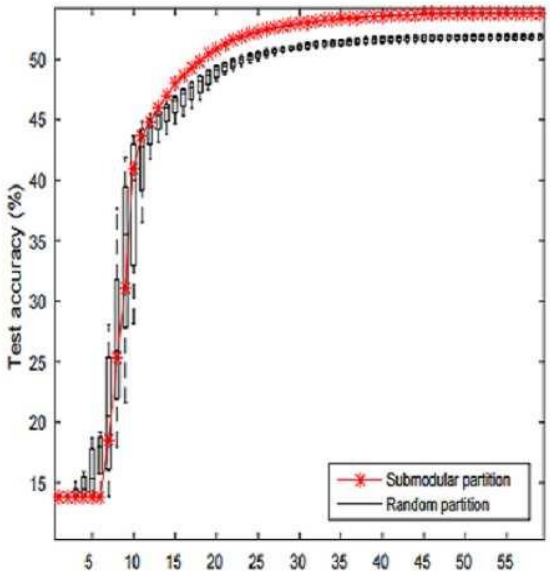

Fig. 8: Accuracy Comparison of Distributed Deep Neural networks after Random Sampling and Submodular Partitioning.

Also, to accumulate Action Rules for the whole input, it requires a separate strategy to combine all the results from multiple nodes such that the final set of Action Rules (in terms of quality and quantity) is better or equal to that of the results from the current action rule extraction approaches running in a single machine. On the other hand, the entire distributed action rule extraction algorithm can take a new dimension of using graph structure like GraphLab [31] or TensorFlow [33] or Parameter Server [34] such that the worker nodes work on their own partition of data and they can communicate with their master for sharing and updating global parameters. Currently there are some on-going research to efficiently use graph structures in Spark [15] particularly for machine learning. This makes graph implementation in Spark much easier than on other frameworks. Special approaches like the algorithms in [7] [8] [9] [10] [11] [12] also can help generating Action Rules in a distributed configuration, such that the performance (time efficiency) of the new algorithm is better than that of current algorithms. Apart from developing a new technique of action rule extraction in a distributed environment, the new extraction technique needs a lot of analysis or validations with results and performances of the current system for multiple real world scenarios. Most importantly, reliability and load balance tests are crucial for distributed algorithm. Furthermore, the new approach needs many use cases on when to use it, when not to use it and what kind of problems are suitable for it to extract Action Rules. Because for some problems, it is better to run algorithms in a single machine instead of multiple computers in a cluster.

8. Conclusion

Action Rules mining is useful for discovering actionable patterns in datasets from several domains such as: medical, financial, industrial, and educational. Actionable patterns extracted from the data are crucial for solving problem these domains. Currently, Action Rules applications in medical [46] and business [47] fields are very important, because the data size grows innumerably every day. There has been active research involvement during past decade to extract Action Rules on datasets in these fields. Multiple methods both in rule-based and object-based approaches have been proposed to extract Action Rules efficiently. Each method has their own style of extraction and fabricating knowledge from the information system. Nowadays, in the epoch of Big Data, where resources like social media and IoT are becoming dominant and fast in providing data, the growth of immense amount data is inevitable. Running even the most efficient Action Rules algorithms on a single machine for such a huge data is a tedious task and time-consuming task.

Cloud Computing is becoming crucial for the better performance of many machine learning algorithms over a past few years to handle rapid generation of big data in distributed and parallel fashion. Action Rules extraction using distributed computing frameworks [14] [15] [26] and publicly available Cloud platforms [17] [18] [16] would be a beneficial upgrade to the current Action Rule mining algorithms. Providing a proper algorithm design for Action Rule mining in distributed environment would allow many applications to benefit from extracting Action Rules in a time efficient manner with large volumes of data.

References

[1] Apache Projects - Pig, Hive, Kafka, SAMOA, [Online]. Available: https://projects.apache.org/projects.html

[2] A. Hajja, A. Wieczorkowska, Z. Ras, "Hierarchical object-driven action rules," in *Journal of Intelligent Information Systems*, vol. 42, no. 2, pp. 207-232, 2014.

[3] A. Hajja, A. Wieczorkowska, Z. Ras, R. Gubrynowicz, "Object-driven action rules and their applications to hypernasality treatment," in *Proceedings of ECML-PKDD workshop on new frontier in mining complex patterns*, Bristol, 2012.

[4] Z. Ras, A. Dardzinska, L. Tsay and H. Wasyluk, "Association Action Rules," in *IEEE International Conference on Data Mining Workshops*, 2008.

[5] Z. Ras, E. Wyrzykowska, H. Wasyluk, "ARAS: Action rules discovery based on Agglomerative Strategy," in *Mining Complex Data, Post-Proceedings of 2007 ECML/PKDD Third International Workshop*, vol. 4944, pp. 196-208, 2008.

[6] L. Tsay, Z. Ras, "Discovering E-Action Rules from Incomplete Information Systems," in *Proceedings of IEEE International Conference on Granular Computing*, Atlanta,Georgia, 2006.

[7] W. Dai, W. Ji, "A MapReduce Implementation of C4.5 Decision Tree Algorithm," in *International Journal of Database Theory and Application*, vol. 7, no. 1, pp. 49-60, 2014.

[8] V. Kolias, C. Kolias, I. Anagnostopoulos, E. Kayafas, "RuleMR: Classification Rule Discovery with MapReduce," in *Proceedings of 2014 IEEE International Conference on Big Data*, 2014.

[9] X. Lin, "MR-Apriori: Association Rules algorithm based on MapReduce," in *Proceedings of 5th IEEE International Conference on Software Engineering and Service Science*, 2014.
[10] V. Nikam, B. Meshram, "Parallel and Scalable Rules Based Classifier using MapReduce Paradigm on Hadoop Cloud," in *International Journal of Advanced Technology in Engineering and Science*, vol. 2, no. 8, pp. 558-568, 2014.
[11] H. Qiu, R. Gu, C. Yuan, Y.Huang, "YAFIM: A Parallel Frequent Itemset Mining Algorithm with Spark," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW '14)*, Washington, DC, USA, 2014.
[12] S. Rathee, M. Kaul, A. Kashyap, "R-Apriori: An Efficient Apriori based Algorithm on Spark," in *Proceedings of the 8th Workshop on Ph.D. Workshop in Information and Knowledge Management (PIKM '15)*, New York, USA, 2015.
[13] G. Wu, H. Li, X. Hu, Y. Bi, J. Zhang, X. Wu, "MReC4.5: C4.5 Ensemble Classification with MapReduce," in *Proceedings of the 4th ChinaGrid Annual Conference (ChinaGrid'09)*, 2009.
[14] J. Dean, S. Ghemawat, "MapReduce: Simplified Dataprocessing on large clusters," in *Proceedings of the 6th conference on Symposium on Operating Systems Design and Implementation*, Berkeley, CA, USA, 2004.
[15] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, I. Stoica, "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)*, Berkeley, CA, USA, 2012.
[16] Microsoft Azure HDInsight, [Online]. Available: https://azure.microsoft.com/en-us/services/hdinsight
[17] Amazon EMR in Amazon Web Services (AWS), [Online]. Available: https://aws.amazon.com/emr
[18] Google Cloud Platform, [Online]. Available: https://cloud.google.com
[19] V. Vavilapalli, A. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed and E. Baldeschwieler, "Apache Hadoop YARN: yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing (SOCC '13)*, New York, USA, 2013.
[20] D. Borthakur, *HDFS Architecture Guide*, Hadoop Apache Project 53, 2008.
[21] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. Franklin, R. Zadeh, M. Zaharia, A. Talwalkar, "Mllib: Machine learning in apache spark," in *Journal of Machine Learning Research*, pp. 1-7, 2016.
[22] G. Nishant, *Apache Kafka*. Packt Publishing Ltd., 2013.
[23] C. Olston, B. Reed, U. Srivastava, R. Kumar, A. Tomkins, "Pig latin: a not-so-foreign language for data processing," in *Proceedings of the ACM SIGMOD international conference on Management of data (SIGMOD '08)*, New York, USA, 2008.
[24] Y. Huai, A. Chauhan, A. Gates, G. Hagleitner, E. Hanson, O. O'Malley and e. al, "Major technical advancements in apache hive," in *Proceedings of the ACM SIGMOD international conference on Management of data*, 2014.
[25] M. Vora, "Hadoop-HBase for large-scale data," in *International Conference on Computer science and network technology (ICCSNT)*, 2011.
[26] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, "Apache Flink: Stream and Batch processing in a single engine," in *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 36, no. 4, 2015.
[27] A. Jain and A. Nalya, *Learning Storm*. Packt Publishing Ltd., 2014.
[28] G. Morales and A. Bifet, "SAMOA: scalable advanced massive online analysis," in *Journal of Machine Learning Research (JMLR)*, pp. 149-153, 2015.
[29] R. Bekkerman, M. Bilenko, J. Langford and et.al, *Scaling up Machine Learning:Parallel and Distributed Approaches*, Cambridge University Press, 2011.
[30] S. Owen, R. Anil, T. Dunning, E. Friedman, *Meet Apache Mahout*, in Mahout in Action, pp. 1-5, 2012.
[31] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, J. Hellerstein, "Distributed GraphLab: a framework for machine learning and data mining in the cloud," in *Proceedings of the VLDB Endowment*, 2012.
[32] J. Wei, K. Chen, Y. Zhou, Q. Zhou and J. He, "Benchmarking of Distributed Computing Engines Spark and GraphLab for Big Data Analytics," in *IEEE Second International Conference on Big Data Computing Service and Applications (BigDataService)*, 2016.
[33] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, et.al, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," in *arXiv:1603.04467*, 2016.

[34] M. Li, D. Andersen, J. Park, A. Smola, A. Ahmed, V. Josifovski, J. Long, E. Shekita, B. Su, "Scaling Distributed Machine Learning with the Parameter Server," in *Proceedings of the 11th USENIX Symposium on Operating System Design and Implementation*, Broomfield, CO, 2014.
[35] Q. Zhang, L. Cheng, R. Boutaba, "Cloud computing: state-of-the-art and research challenges," in *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 1-7, 2010.
[36] IBM Bluemix, [Online]. Available: https://www.ibm.com/cloud-computing/bluemix
[37] K. Adhatrao, A. Gaykar, A. Dhawan, R. Jha, V. Honrao, "Predicting Student's Performance using ID3 and C4.5 Classification algorithms," in *International Journal of Data Mining and Knowledge Management Process (IJDKP)*, vol. 3, pp. 39-52, 2013.
[38] L. Breiman, "Bagging Predictors," in *Machine Learning*, vol. 24, pp. 123-140, 1996.
[39] R. Agrawal, R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," in *20th International Conference on VLDB*, 1994.
[40] A. Srinivasan, T. Faruquie, "Data and task parallelism in ILP using MapReduce," in *Machine Learning*, vol. 86, no. 1, pp. 141-168, 2012.
[41] J. Grzymała-Busse, S. Marepally, Y. Yao, "An Empirical Comparison of Rule Sets Induced by LERS and Probabilistic Rough Classification," in *Rough Sets and Intelligent Systems*, vol. 1, pp. 261-276, 2013.
[42] J. Rauch, M. Šimůnek, "Action rules and the GUHA method: Preliminary considerations and results," in *International Symposium on Methodologies for Intelligent Systems*, pp. 76-87, 2009.
[43] K. Wai, R. Iyer, J. Bilmes, "Submodularity in Data Subset Selection and Active Learning," in *International Conference on Machine Learning (ICML)*, 2015.
[44] K. Wei, R. Iyer, S. Wang, W. Bai and J. Bilmes, "How to Intelligently Distribute Training Data to multiple computer nodes: Distributed Machine Learning via submodular partitioing," in *Neural Information Processing Society (NIPS) workshop*, Montreal, Canada, 2015.
[45] K. Wei, R. Iyer, S. Wang, W. Bai, J. Bilmes, "Mixed Robust/Average Submodular Partitionaing: Fast Algorithms, Guarantees and Application," in *Advances in Neural Information Processing Systems*, 2015.
[46] M. Almardini, A. Hajja, L. Clover, D. Olaleye, Y. Park, J. Paulson and Y. Xiao, "Reduction of Hospital Readmissions Through Clustering Based Actionable Knowledge Mining," in *IEEE/WIC/ACM International Conference on Web Intelligence (WI'16)*, 2016.
[47] J. Kuang, Z. W. Ras, A. Daniel, "Hierarchical Agglomerative Method for Improving NPS," In: *Kryszkiewicz M., Bandyopadhyay S., Rybinski H., Pal S. (eds) Pattern Recognition and Machine Intelligence. PReMI 2015*, Lecture Notes in Computer Science, vol. 9124. Springer, Cham, 2015.