DISTRIBUTED ACTIONABLE PATTERN MINING

by

Arunkumar Bagavathi

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Computing and Information Systems

Charlotte

2019

Approved by:

Dr. Angelina A. Tzacheva

Dr. Zbigniew W. Ras

Dr. Jing Yang

Dr. Maciej Noras

ABSTRACT

ARUNKUMAR BAGAVATHI. Distributed Actionable Pattern Mining. (Under the direction of DR. ANGELINA A. TZACHEVA and DR. ZBIGNIEW W. RAS)

In recent years, the internet has become faster, computer storage has become larger, data collected by organizations as well as web and social media has grown tremendously. Data Mining tools require adaptations to cope with analyzing massive amounts of data. Ecosystems like Hadoop, MapReduce, Spark, and other cloud platforms have emerged to store, manage, and process large data in reasonable time. Recent work has adapted certain machine learning tools to run on these systems. However, many rule extraction data mining tools still lack adaptation or software packages for processing on cloud platforms, which presents a challenging problem. Actionable Pattern Mining is a type of rule extraction data mining approach for discovering actionable knowledge, that the user can utilize to their advantage. Traditional classification rules predict a class label of a data object. In contrast, Action Rules produce actionable knowledge or suggestions on how an object can change from one class value to another more desirable one. In this work, we discuss association and classification rule mining algorithms with distributed environments. We focus on Actionable Pattern Discovery and propose several approaches to adapt this method for processing in a distributed environment, and extract actionable patterns from big data using distributed computing frameworks. We experiment with several datasets, including Car Manufacturing, Mammographic Mass, Charlotte Business Wise data, Net Promoter Score business data, and Hospital Readmission data. Results show that rule mining algorithms can successfully adapt to cloud computing environment, in order to scale and handle big data. We show that such an adaptation improves the execution time efficiency of rule mining algorithms.

DEDICATION

To my family and friends with love...

ACKNOWLEDGEMENTS

First and foremost, I thank my advisor Dr. Angelina A. Tzacheva, who encouraged me to join the Ph.D. program at UNCC and helped me in many ways through out my Ph.D. to mold my research and interpersonal skills. I also thank my co-advisor Dr. Zbigniew W. Ras for supporting and motivating me all the time to come up with new ideas and problems. I also thank Dr. Siddharth Krishnan for acting as my mentor for the past two years. He has gone above and beyond in guiding my development as researcher. He also provided me numerous computing resources to conduct my research. I would further thank my dissertation committee Dr. Jing Yang and Dr. Maciej Noras for inspiring me with innovative ideas and helped creating an effective research.

I thank my colleagues and collaborators: Mamoun Al-Mardini, Katarzyna Tarnowska, Jaishree Ranganathan, Abhishek Tripathi, Sharath Suryanarayanaprasad, Lavanya Ayila, Aabir Kumar Dutta, Varun Rao, Apurwa Apurwa for adding technical innovations to my research study and helping with all tedious experiments and analysis.

I would also thank the IT support team: Jon Halter, Chuck Price and Joe Matesich for their enormous effort in setting up and maintaining the hadoop cluster for research purposes. I appreciate their support for resolving all kinds of issues that I confronted during the research.

The Department of Computing and Information Systems is a tremendous place for graduate students. Apart from extraordinary professors and peers, the student support from the department is priceless. I thank Dr.Bojan Cukic for providing funds to purchase HCUP datasets and for conference travels. I also thank Crystal Green, Jon Rea, and Erica Naylor for being efficient in several travel reimbursements and notifications.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Knowledge Discovery is a series of steps to be pursued on the given data. It involves cleaning and transforming the given large data to an appropriate format, apply data mining algorithms to extract knowledge from the data and finally evaluate and interpret results[1]. Data mining consists of multiple concepts, to obtain knowledge from the data, such as rule extraction, classification, regression and clustering. These techniques produce tremendous amount of results or patterns, regardless of whether they are of user's interest, or not. The primary obstacle for much of the data mining and machine learning algorithms is *the lack of actionability* [2].

1.1 Actionable Knowledge

Actionable Pattern Mining is a rule based knowledge discovery system that discovers actionable patterns from the data. Most of the data mining or machine learning techniques learn a model from the data and next make a prediction for the new data. However, they do not provide any actionable insights on the given data. The statement which says: 'Knowledge is Power' may not be true, the statement: 'Used Knowledge is Power' is more appropriate [3].

For example, a car manufacturing company can predict its profit using some data mining or machine learning algorithm. However, results of this algorithm, do not provide any insight on how the company can improve its profits. In contrast, Actionable Pattern mining may suggest Actions the company can undertake in order to increase its profit, or to accomplish its goals. Of course, there is a cost they have to spend to achieve recommended action [4].

1.2 Interestingness Measures

A shortcoming of data mining is that the large quantity of discovered patterns due to the massive availability of data. These patterns have to be analyzed by some post-processing approaches in order to minimize their number. In this work we only focus on actionable patterns that have a form of *IF-THEN* rules. One of the main motivations in post-processing analysis is to simplify discovered patterns in order to improve knowledge comprehensibility to the users. Also, because the quantity of patterns discovered by a data mining algorithm is large, we want a subset of these rules, which are interesting, because the existing knowledge discovery algorithms discover accurate information rather than interesting ones. There are two aspects of interestingness that have been studied in data mining literature, objective and subjective measures

Objective measures are data-driven and domain-independent. Generally, these measures evaluate the resulting rules based on quality as well as the similarity between them, rather than considering what the user believe about domain.

Subjective measures by contrast, are user-driven and domain-dependent. For example, the user may be involved to specify rule template, indicating which attribute(s) must occur in the rule to be interesting from his/her point of view [5].

1.3 Big Data

The towering production of data in recent years captured by organizations, which makes heavy use of social media, multimedia and Internet of Things (IoT), has led to the current era of big data. These organizations are collecting large amounts of data from their daily routines and their data size grows exponentially every year. Data is pouring at a massive rate setting the problem of analyzing the big data as

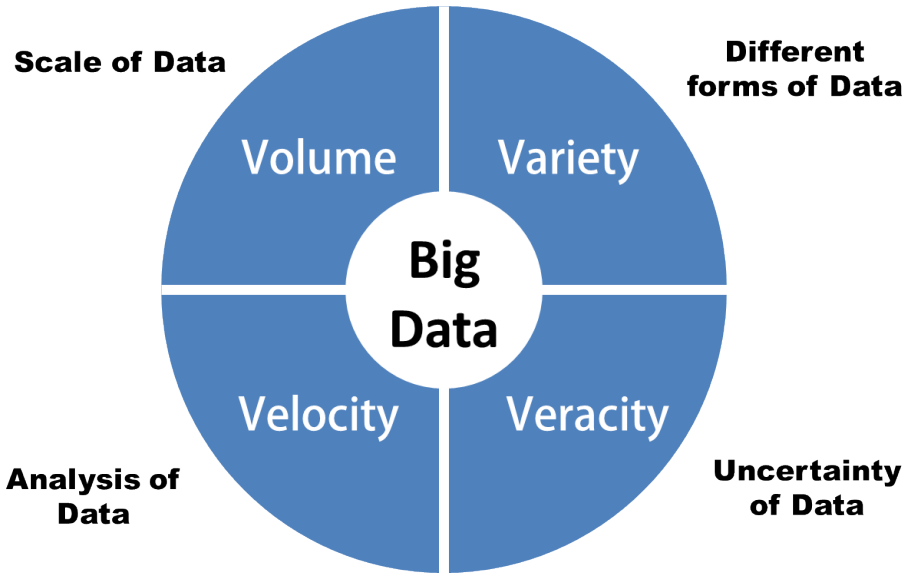


Figure 1.1: Varieties of problems in handling Big Data. These problems are categorized into Volume, Variety, Velocity and Veracity

a modern trend [6]. As given in Figure 1.1, the topic of big data covers Volume, Variety, Velocity and Value [7].

When the incoming data is at great velocity and when the data reaches a massive size, traditional data mining and machine learning methods are no longer capable to process the data and proves difficult to obtain results in a reasonable time. This problem applies to Association Rule Mining, Classification Rule Mining, and of course, Actionable Pattern Mining [8]. Thus, it requires high computational power to uncover hidden values from such a large data. One idea to deal with the big data is that to distribute the data to multiple machines and do data processing on each of them. This require us to setup a cluster of computers, configure each computer with required softwares, manage communication between computers in the cluster and provide security for such clusters.

1.4 Cloud Computing

Cloud Computing is an ubiquitous, on-demand and easy-to-use model that provide access to a pool of computing resources such as servers, storage, softwares, security and services [9]. Cloud computing comprise of different types of services, which users can make use for their requirements [10]. Figure 1.2 give several types of services provided by service providers. These services include *Software-as-a-Service(SaaS)* - providing softwares for use, *Platform-as-a-Service(PaaS)* - providing requied platforms like IDEs to develop programs and web severs to host web applications and *Infrastructure-as-a-Service(IaaS)* - providing a configurable computing resource(s) like virtual machines. Many applications are currently being deployed in cloud and may continue to increase due to lack of availability of computing resources such as good computer configuration and data storage for the big data that would be consumed by the developed applications [11]. Few cloud service providers like Amazon Web Service (*AWS*) [12], Microsoft Azure [13], Google Cloud [14] and IBM Cloud [15] started providing all required services for data scientists to tackle the problem of big data.

1.5 Big Data Analysis in Cloud : Distributed Computing

Cloud Computing and Big Data work co-jointly in recent years. Due to the increasing availability of commercial Cloud platforms, one can easily do distributed computing for performing data analysis on the big data in a distributed fashion. Cloud computing architecture can serve useful for the problem of big data storage and analysis [16]. Very large datasets can be sliced into small chunks and stored in multiple computers in the cluster. Other benefits of using cloud computing infrastructure is that the results can transfer between different locations efficiently within the cluster and since the data processing algorithms works only on a small portion

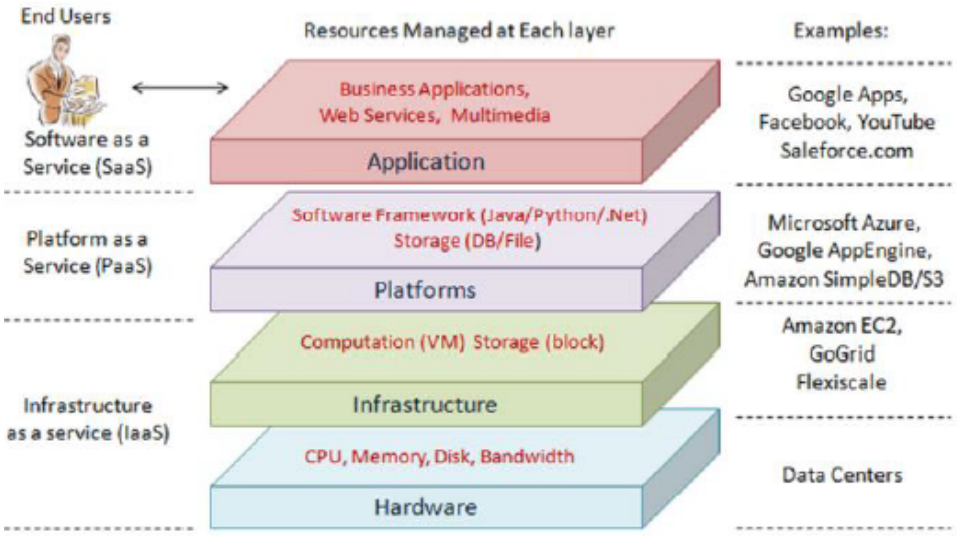


Figure 1.2: Basic services in Cloud Computing for application developers, data scientists and researchers

of the data, we get better performance on the processing algorithms. For processing this big chunk into multiple locations various cloud computing frameworks have to coordinate with each other, swhich increases the complexity of big data processing [17]. Many distributed big data processing models such as MapReduce [18] and Spark [19] are some of the good examples that use cloud computing technologies efficiently for analyzing massive data stored in the cluster.

1.6 Research Focus : Actionable Pattern Mining using Cloud Environment

Actionable Patterns prove to be helpful in multiple applications like improving customer satisfaction in the business sector [20], reducing patient readmission in hospitals in medical sector [21], in determining actionable patterns in hoarseness disease [22] and in music recommendations [23]. Even though, it is being used in multiple scenarios, the primary snag of these applications is that they use traditional way of extracting action rules in a standalone computer.

Consequently, number of research work has started to focus on distributed methods for machine learning algorithms. However, limited work exists on adapting Associa-

tion Rule Mining, and Classification Rule mining to a distributed environment. And very few papers exist on adapting Actionable Pattern mining to a distributed environment [8]. There are no commercial, or open-source packages available for Actionable Pattern mining in a distributed environment. In this work, we propose several methods to adapt extracting Actionable patterns in a form of Action Rules using cloud based distributed processing frameworks. We build a software package, which we plan to make available to popular open-source tools, such as Hadoop, and Apache Spark Machine Learning library (MLLib). In addition we explore the concept of Cost of Action Rules, in order to perform the recommendation actions. We propose a new method of extracting Action rules of lowest cost in a distributed environment.

The rest of the paper is organized as follows: in Chapter 2, we provide background knowledge on rule-based algorithms to extract knowledge from the data and distributed computing frameworks ; Chapter 3 describe existing algorithms to extract actionable patterns; Chapter 4 give an overview of the state-of-the-art methods for distributed rule mining and detail our proposed methods to extract actionable patterns in distributed frameworks; Chapter 5 describe our proposed methods for extracting cost efficient actionable knowledge; Chapter 6 describe the experiment and results using datasets in 4 different domains; in Chapter 7 we conclude; and finally in Chapter 8 we discuss challenges, and provide directions for future.

CHAPTER 2: BACKGROUND

In this section, we provide background information about basic concepts such as: rule-based data mining techniques, association rules, decision rules and Action Rules, as well as the environment to extract Action Rules. We describe the properties to evaluate Action Rules including Support, Confidence, Utility, and Cost and Feasibility. We provide an overview of existing distributed distributed data processing frameworks, and their scalability for very large datasets, as well as and their benefits for Machine Learning.

2.1 Rule-based Learning

In this research, we primarily focus on rule-based data mining. Rule-based learning or data mining identifies rules to store and manipulate knowledge from the data. Rule-based learning differs from other machine learning algorithms, which identify a common model that can be utilized in the future to make predictions. Rules takes the format as given in Equation 2.1, where the *antecedent*(left side of the rule) is a conjunction of conditions and the *consequent* (right side of the rule) is a resulting pattern for the conditions in antecedent.

$$condition(s) \rightarrow result(s) \tag{2.1}$$

2.2 Information Systems

Information systems roots from the concept of rough sets [24]. Information system is a set of objects and attributes as given in Equation 2.2. This equation represents that an information system S_i can take a set of objects U and an infinite set of at-

tributes A, where each attribute in A can contain a finite set of values V_a.

$$S_i = (U, A); a : U \rightarrow V_a, \forall a \in A \tag{2.2}$$

2.3 Association Rules

Association Rules are widely used to derive correlations, associations and frequent patterns from data objects of a given data [25]. They find applications in multiple domains such as risk management, shopping, etc. One of such most popular applications of association rules is the *Market Basket Analysis*. It examines patterns of customer shopping in a grocery store. For example, association rules can recommend to a grocery store manage that, if a customer purchase milk, bread and diaper together, they purchase beer also. Thus the manager can put some offers or deals on beer products. On the other hand, this helps in sequential relationships also. For example, association rules can determine if a customer purchase a television and a gaming console during this year's black friday sales, it is more likely they purchase a home theater next year. Apriori algorithm [26] is the well known algorithm to extract association rules. The algorithm starts with *1-item* and loops until *k-items* combinations.

2.4 Decision Tables

Decision Tables mark a special case of the Information Systems [27]. In case of decision tables, the attribute space in the Information Systems A can be either conditional attributes or a decision attribute (d) and the condition attributes in turn can be categorized into either stable (A_{st}) or flexible attributes (A_{fl}). Thus, decision table in terms of an information system can take a representation given in Equation 2.3

$$S = (U, A_{st}, A_{fl}, d) \tag{2.3}$$

From Equation 2.2 and Equation 2.3, we can depict that the information system S_i can be converted into a decision table S by having all list objects U as such and splitting the attributes A as $A = (A_{st} \cup A_{fl} \cup d)$, where A_{st} is a set of stable attributes, A_{fl} is a set of flexible attribute and d is a decision attribute. The special case of the decision attribute d is that it should be of flexible attribute type but does not belong in the set of flexible attributes A_{fl}. Flexible attributes can change their values into another value of the same attribute whereas stable attributes remain constant once assigned. Some examples of stable attributes in the real world scenario are *Date of Birth* and *Zipcode*

Table 2.1: Sample Decision Table

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

Table 2.1 gives a sample decision table. From Table 2.1, if we consider attribute B as a stable attribute, and attribute D as a decision attribute, this table can match with Equation 2.3 as given in Equation 2.4

$$S = (\{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}, \{B\}, \{A, C\}, D) \tag{2.4}$$

2.5 Decision Rules

A decision rule from the given decision table like the one given in Table 2.1 can take a representation of Equation 2.1. The *antecedent* of a rule can be a singleton

condition or assumption or it can be a conjunction of conditions like $c_1 \cap c_2 \cap \ldots \cap c_n$, and the *descendant* is a singleton result based on a decision attribute D. A decision rule classifying a group of conditions to a decision d_i means that data objects which satisfy (match) the rule's antecedent, go into the d_i category.

In terms of prediction or classification problem, decision rules from the training data are useful in classifying data objects to one of the decision attribute values d_i. New data objects come with a same attributes as given with the training data. Thus decision rules help in a classification problem to classify new objects to one of the decision values based on their conditions(*descendant*) over the data attributes.

2.6 Action Rules

We focus on a special type of rule based technique named as 'Action Rules' [27]. Action Rules recommend possible transitions of data from one state to another, which the user can use to their advantage. In other words, Action Rules helps to reclassify the results of a classification algorithm from one category to another, recommending patterns to improve performance of an object or establishing better work to the user. Some of the applications for Action Rules are: improving customer satisfaction in business [20] and reducing hospital readmission in the medical field [21]. For Action Rules extraction algorithms, the attributes in the data can be split into *Stable Attributes* and *Flexible Attributes* along with the *Decision attribute* which is the final decision that the user need to achieve. Stable attributes in any Action Rule AR remain constant or cannot form action in AR. While flexible attributes can change their value from a_i to a_j. Action Rules can take the representation as given in Equation 2.5, where Ψ represents a conjunction of stable features, $(\alpha \rightarrow \beta)$ represents a conjunction of changes in values of flexible features and $(\theta \rightarrow \phi)$ represents desired decision action.

$$[(\Psi) \wedge (\alpha \rightarrow \beta)] \longrightarrow (\theta \rightarrow \phi) \tag{2.5}$$

Action Rules of the form given in Equation 2.5 give recommendations to users that they can achieve the desired action $(\theta \rightarrow \phi)$, given on the *consequent* part of the rule, if they perform a series of actions $[(\Psi) \wedge (\alpha \rightarrow \beta)]$, given on *antecedent* part of the rule. Action Rules are validated using Support, Confidence, Utility and Coverage measures.

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. As given in Equation 2.5, Action Rules have 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 a set of 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.

Consider a decision table as given in Equation 2.3. *Action term* or *Atomic Action Term* can be given by the expression of $(m, m_1 \rightarrow m_2)$, where $m \in (A_{fl} \cup A_{st})$ and m_1, m_2 are the values of the attribute m. $m_1 = m_2$ if $m \in A_{St}$. In that case, we can simplify the expression as (m, m_1) or $(m = m_1)$. Whereas, $m_1 \neq m_2$ if $m \in A_{Fl}$

Action Rules can take a form of $t_1 \cap t_2 \cap .... \cap t_n$, where t_i is an atomic action term or action term and the Action Rule is a conjunction of action terms to achieve the desired action based on the given decision attribute d. Example Action Rule for the Decision table given in Table 2.1 is given in Equation 2.6 below. The below equation represents that the user can obtain the relevant action $(D, d_1 \rightarrow d_2)$, if they do actions $(A, a_1 \rightarrow a_2)$ and (B, b_1).

$$(A, a_1 \rightarrow a_2) \wedge (B, b_1) \longrightarrow (D, d_1 \rightarrow d_2) \tag{2.6}$$

2.7 Objective Measures of Action Rules

Action Rules are considered valuable to a user, if they find the rules as interesting. In all our approaches we use the following objective measures of extracted actionable knowledge: support, confidence, utility, and coverage[28]. The higher these values, the more interesting the rules are to the end user. More measures of objective interestingness include R-interestingness, intensity of implication, discrimination, simplicity, certainty, statistical significance, accuracy, J-measure, certainty, RI, strength and disjoint size, imbalance of the class distribution, attribute cost, misclassification cost, and asymmetry of classification rules.

Consider an action rule R of form given in Equation 2.7

$$R : (Y_1 \rightarrow Y_2) \longrightarrow (Z_1 \rightarrow Z_2) \tag{2.7}$$

where,

Y is the *antecedent* or condition part of R

Z is the *subsequent* or decision part of R

Y_1 is a set of all left side action terms in the condition part of R

Y_2 is a set of all right side action terms in the condition part of R

Z_1 is the decision attribute value on left side

Z_2 is the decision attribute value on right side

The support and confidence of action rule R as represented in Equation 2.7 are given in Equation 2.8 and Equation 2.9 respectively.

$$Support(R) = min\{card(Y_1 \cap Z_1), card(Y_2 \cap Z_2)\} \tag{2.8}$$

$$Confidence(R) = [\frac{card(Y_1 \cap Z_1)}{card(Y_1)}] \cdot [\frac{card(Y_2 \cap Z_2)}{card(Y_2)}] \tag{2.9}$$

Tzacheva et.al [29] proposed a new set of formula for calculating Support and Confidence of Action Rules of the type given in Equation 2.7. Their idea is to reduce complexities in searching the data several times for Support and Confidence of an Action Rule. The new formula are given below in Equation 2.10 and 2.11.

$$Support(R) = \{card(Y_2 \cap Z_2) \tag{2.10}$$

$$Confidence(R) = [\frac{card(Y_2 \cap Z_2)}{card(Y_2)}] \tag{2.11}$$

Tzacheva et. al [29] also introduced a notion of utility for Action Rules. Utility of Action Rules takes a following form. For most of cases Utility of Action Rules equals the Old Confidence of the same Action Rule. The formula for calculating *Utility* of action rule R is given in 2.12

$$Utility(R) = [\frac{card(Y_1 \cap Z_1)}{card(Y_1)}] \tag{2.12}$$

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

2.8 Subjective Measures of Action Rules - Cost and Feasibility of Action Rules

Subjective measures are calculated depending on the subject matter expertise of a person who examines the patterns such as actionability and unexpectedness. Subjective measures have been studied in the literature using *unexpectedness*, and *actionability* [30]. A rule is **unexpected** if it contradicts the expert belief on a domain and therefore surprising. A rule is **actionable** if the user can take any action to his/her

advantage based on this rule. In this work, we measure subject measure in the form of cost of the actionable knowledge.

Author Dardzinska introduced the notion cost and feasibility of an Action Rule in [31]. Assume that S is an information system. Let $b \in B$ is flexible attribute and b_1, b_2 are values of b. $\rho(b_1, b_2)$ mean any number from the open interval $(0,1) \cup \{+\infty\}$ which describes the cost to change the value from b_1 to b_2 by the user of the information system S.

The value of $\rho(b_1, b_2 \approx 0)$ is interpreted that the change of values from b_1 to b_2 is quite trivial.

The value of $\rho(b_1, b_2 \approx 1)$ is interpreted that the change of values from b_1 to b_2 is very difficult to be achieved.

The value of $\rho((b_1, b_2 \approx +\infty)$ is interpreted that the change is not feasible.

Also, if $\rho(b_1, b_2) < \rho(b_3, b_4)$, then change of values from b_1 to b_2 is more feasible than the change from b_3 to b_4.

The values $\rho((b_i, b_j)$ are given by the user of information system and they should be seen as atomic values needed to introduce the notion of the feasibility of an Action Rule.

Assume now that equation (2.13) is a (r_1, r_2) Action Rule [32].

$$\begin{aligned} r = [(b_1, v_1 \to w_1) \cap (b_2, v_2 \to w_2) \cap ... \cap \\ (b_m, v_m \to w_m)](x) = (d, d_1 \to d_2)(x) \end{aligned} \tag{2.13}$$

By the cost of rule r denoted by $\wp_c$ means value in equation (2.14)

$$cost(r) = c \tag{2.14}$$

Rule $\big(r\big)$ is feasible $\big(cost(r) \leq \wp_c\big(d_1, d_2\big)$ if which means that $cost(r)$ has to be a finite number and the cost of the conditional part of the rule has to be a finite number and

the cost of the conditional part of the rule has to be lower than the cost of the decision part of the rule.

Consider d is a decision attribute, assume that $D_s[(d, d_1 \rightarrow d_2)]$ denotes the set of all Action Rules in S having the term $(d, d_1 \rightarrow d_2)$ on the decision site. Among all Action Rules in $D_s[(d, d_1 \rightarrow d_2)]$ we have to choose a rule with the smallest cost value. However it can still happen that the rule we chose has the cost value not acceptable by the user of the information system S. The cost of the Action Rule in equation (2.15) might be high only because the cost value of one of its sub-terms in the conditional part of the rule is high.

$$\begin{aligned} r = [(b_1, v_1 \rightarrow w_1) * (b_2, v_2 \rightarrow w_2) * ...* \\ (b_m, v_m \rightarrow w_m)](x_i) \rightarrow (d, d_1 \rightarrow d_2)(x_i) \end{aligned} \tag{2.15}$$

In the real world, each and every action take some form of cost. For example, an action of a car ($Idle \rightarrow Running$) cost a fraction of fuel and energy to its engine. Similarly every single actionable recommendation appearing in an action rule costs something to its end user. The cost can occur in multiple forms including energy, currency, human effort, etc. Thus the cost of each action in action rules can be assigned only by a domain expert, who has experience in the recommended actions. The end user to whom the action rules are recommended consider using the recommendation, only if the recommendations charge a reasonable cost to them. In other words, cost of action rules measure interestingness of the given recommendations [4]. The extracted actionable patterns to attain the necessary decision action are more interesting to users if they cost less to them. Cost of an Action Rules is one of the heuristic strategies for creating new action rules, where data objects in the decision table S supporting the new Action Rule also supports intial Action Rule but the cost of reclassifying them is lower or even much lower for the new rule [33].

2.9 Meta Actions

Meta-Actions are referred to as higher level concepts that model generalization of Action Rules by authors Touati et al. [34] and Tzacheva et al. [35]. Consider the medical example from section , in order to move a patient from worst prognoses state to good prognoses state requires some treatment procedures to be changed or some medication to be changed. This actionable knowledge is represented by meta-actions. A more formal definition of meta-action given by authors Touati et al. in [34] is as "Meta-actions associated with an information system S are defined as higher level concepts used to model certain generalization of Action Rules. Meta-actions, when executed, trigger changes in values of some flexible attributes in S."

Authors Tzacheva and Ras [35] present a strategy for generating association Action rules and action paths by introducing the use of Meta-actions and influence matrix [36]. According to Tzacheva and Ras [35] some higher-level actions called Meta-actions are required to trigger the change of flexible attributes in order to move undesirable objects into a desirable group. For example, consider an example given by Dardzinska [31]. "The rehabilitation data with classification attributes of : *Do exercises effectively, Stimulate patient during exercises, Provide sufficient feedback about condition of patients.* Examples of meta-actions for this data include : *Change the kind of exercises, Change the orthotic insoles.* The authors use Influence matrix to identify relationship between meta-actions and classification attributes. Similarly, Tzacheva and Ras [35] use Influence matrix to identify which candidate association Action Rule and action paths are valid with respect to meta-actions and hidden correlation between classification attributes and decision attributes. Action paths are a sequence of action terms.

2.10 Rough Sets

Rough sets [37] derive knowledge from data represented in Information System(IS) which takes a form of (U, A), where U is a non-empty finite set of objects and A is a non-empty finite set of attributes such that $a : U \rightarrow V_a, for a \in A$, where V_a is the value set of a. Rough sets are based on intuition that different set of attribute from the Information System yield different concept granulations. Concepts represent a set of entities in the information system *IS* that maintains *indiscernable* relation. With any $B \subseteq A$ fro the information system(IS) there is an equivalence relation given in Equation 2.16, where $IND_{IS}(B)$ is called *B-indiscernibility relation.*

$$IND_{IS}(B) = (x, y) \in U^2 | \forall a \in B, a(x) = a(y) \tag{2.16}$$

A part of U represent a family of all equivalence classes denoted by $[x]_B$. Let $X \subseteq U$ be a set that we want to represent using attribute subset $B \subseteq A$ in the Information System. However, X cannot be represented as it may include and exclude objects that are indiscernible based on B. however, we can approximate X with only information in B by constructing the *B-upper* approximation and *B-lower* approximation of X denoted by B^1X and B_1X respectively, where B^1X and B_1X satisfy conditions given in Equation 2.17 and Equation 2.18 respectively.

$$B^1X = x | [x]_B \subseteq X \tag{2.17}$$

$$B_1X = x | [x]_B \cap X \neq \emptyset \tag{2.18}$$

The boundary region of X, given in Equation 2.19 consists of objects that we cannot classify them into X. A set is a *Rough Set* if the boundary region is non-empty otherwise the set is considered as a *Crisp Set*

$$BN_B(X) = B^1X - B_1X \tag{2.19}$$

2.11 Reducts

One of the important functionalities of rough sets is feature selection by preserving the characteristics of original feature set and deleting redundant information from them [38]. The key idea is to keep only attributes that preserve *indiscernibility* relation and set approximations. By following this idea, it results in several subset of attributes and the minimal from those subsets are called *reducts*.

We believe data Granules such as Rough Sets and Reducts can be used for intelligent division of data, into chunks, which are then provided to each individual worker node on the Cloud Cluster for processing. Next, the Actionable patterns are discovered on each chunk. We plan to analyze the resulting patterns and their properties, and how closely they approximate the original dataset properties if it were not divided before processing.

2.12 Granular Computing and Information Granules

Granular Computing (GrC) is a domain that makes use of information granules for solving complex human-centric problems [39]. The idea of granular computing is widely used in multiple areas like data processing, machine learning, rough set theory, decision trees and artificial intelligence. With the key idea of information granules, Granular Computing can also be used in knowledge representation and data mining. *Information granules* are a collection of granules, where each *granule* is a set of data objects are stacked together based on their similarity, functionality or indistinguishability [40]. Thus a granule can be seen as a subset of a larger problem, that can be used effectively to solve a complex task. Information granulation is the process of breaking a complex object into smaller pieces called information granules. Informa-

tion granulation, thus can solve more complex problems by considering meaningful levels of granularity of the problem [41].

In this work, we plan to divide the data using Granules, before feeding it to individual cluster worker nodes for processing. Granular Computing is a method to abstract the information and represent them as small chunks of knowledge called information granules [42]. Granular computing is a big umbrella to represent several theories, methodologies, tools and techniques to use information granules for problem solving. Granular computing covers popular data mining concepts such as discretization: reducing the resolution of attributes in a data [43], clustering or aggregation : grouping attributes together to reduce the size of the attribute space [44] and rough sets [37].

2.13 Hadoop MapReduce

MapReduce [18] is a computational model that has a potential to distribute the given data and process them at a same time in fault tolerant and scalable approach. The authors claim that the goal of MapReduce is to make users to think about how to do put their algorithm into MapReduce framework. MapReduce also take care of other complex functionalities such as data distribution, parallelization, load balancing and fault-tolerance during node failures. Hadoop MapReduce works on Hadoop Distributed File System(HDFS) [45], a distributed file system used to store the data and access it quickly. Hadoop also prevents data loss by having three copies of each data across the cluster.

MapReduce works with two functions: *Map* and *Reduce*. Both functions take input and produces output as <Key, Value>pairs. Inputs and outputs of these functions are stored in HDFS for quicker access. The whole process is monitored by a *Master node*. *Master node* finds an appropriate number of available nodes at present and assign them as slave nodes to work as *Mappers*. Hadoop [46] preserves data locality by assuring that distance between data node and slave node is minimum. Now the

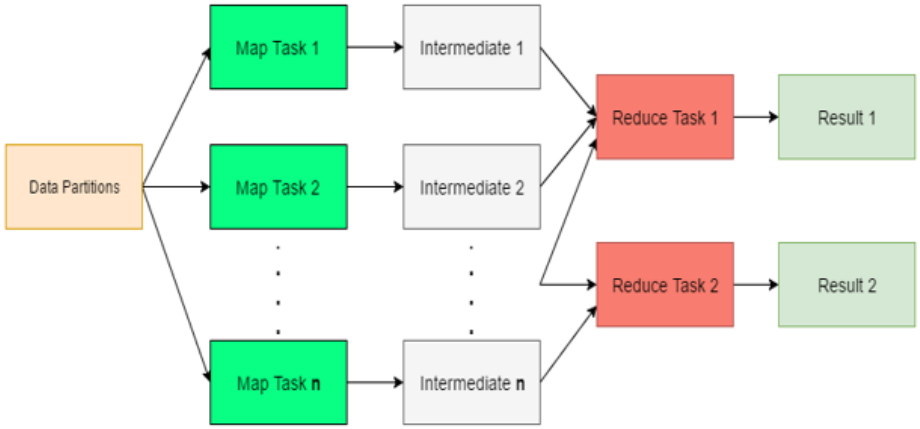


Figure 2.1: Simple steps in MapReduce execution

slave nodes getting their own tasks, MapReduce starts its Map phase on them. Since the map function is made on a small split of a large data, their results are generally considered as intermediate outputs. Each map function writes their output <Key, Value>pairs back into HDFS. The master node again finds an available slave node(s) to perform reduce function and mark them as *Reducers*. Reduce function collects all values for a single key, perform computations on the collection and writes the output as <Key, Value>pair again to the HDFS. Figure 2.1 shows an overview of a basic MapReduce execution. This output can be used again by another MapReduce phase or can be used by another application to perform other computations.

2.14 Apache Spark

Apache Spark [19] is a framework that is similar to MapReduce [18] to process large quantity of data efficiently in a parallel fashion and in a short span of time. The disadvantage of MapReduce framework is frequent system's disk access for writing and reading the data between Map and Reduce phases. However, Spark introduces a distributed memory abstraction strategy named Resilient Distributed Datasets(RDD). The RDDs works by splitting the data into multiple nodes, do in-memory computations on whose nodes and store the results in memory itself if there are any available

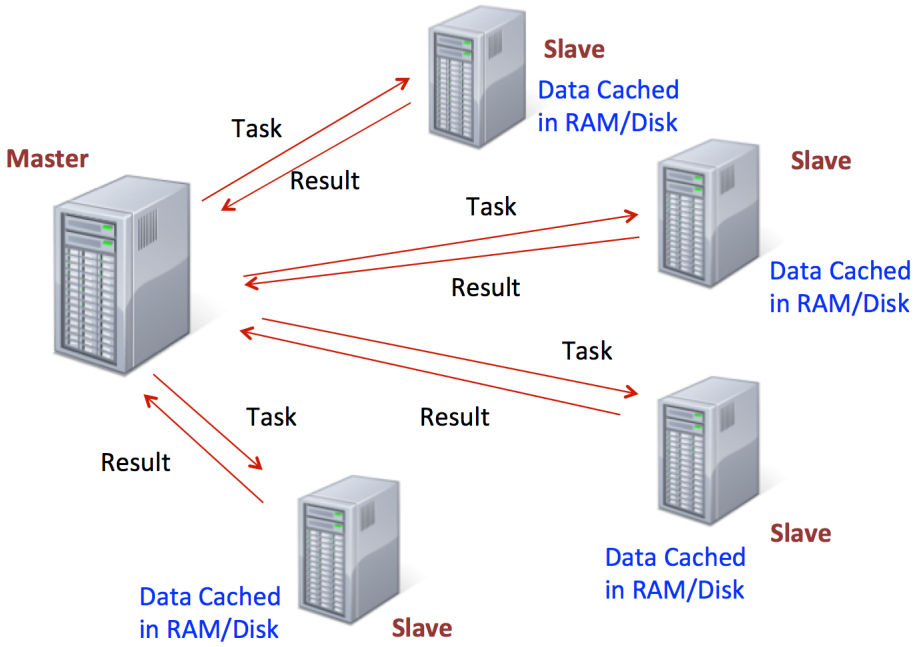


Figure 2.2: Execution of Spark with In-Memory computation and Resilient Distributed Datasets

space in RAM. These results can be accessed for future processes and analyses, which in-turn create another RDD. Once the RAM goes out-of-memory, Spark uses some strategies to push the results that are unused for a long time to the disk. Thus, Spark cuts-off large number of disk accesses for storing intermediate outputs like in Hadoop MapReduce. Spark works in a *Master-Slave* approach. The Driver node(*Master*) allocate tasks to the Worker nodes(*Slaves*). Spark preserves data-locality (i.e) locating worker nodes nearer to the current node which contains a part of the data. A task that the worker perform can be either a *Transformation* or an *Action*. During *Transformation* stage, computations are made on the data split and results are stored in-memory of the worker node. Results of all worker nodes together form another RDD. While the *Action* stage on an RDD collect results from all workers and send it to the driver node or save the results to a storage system. Figure 2.2 shows an overview of the execution of Spark.

Spark helps machine learning algorithms which relies on multiple iterations on the

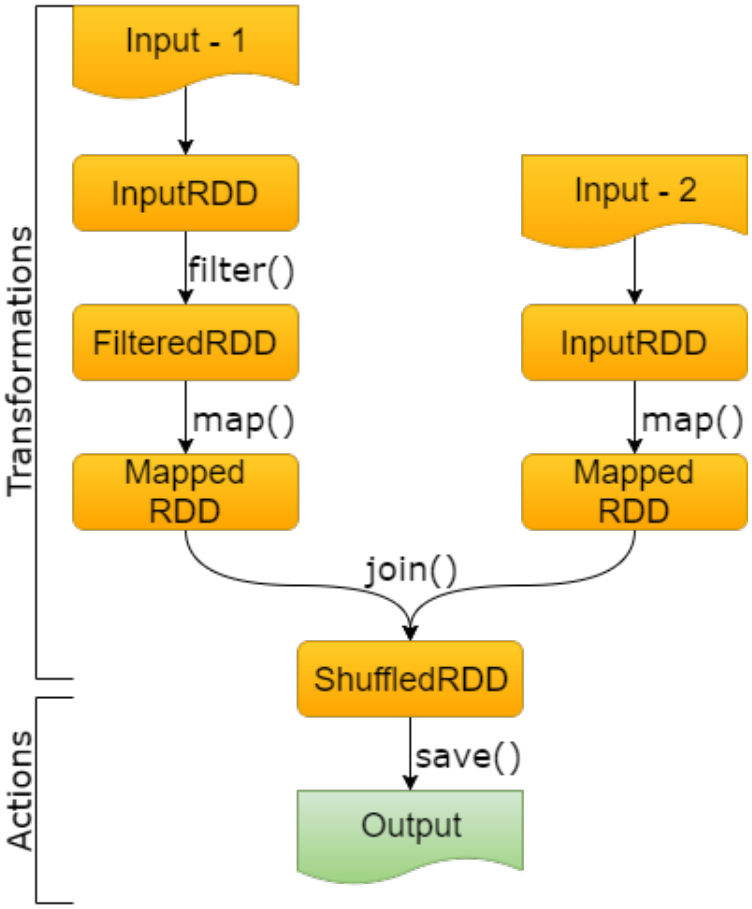


Figure 2.3: Example on how Spark maintains Lineage Graphs when processing Big Data

given data with the help of RDD's in-memory computation. Spark handles node failures by having a lineage graph of RDDs. The lineage graph is a Directed Acyclic Graph(DAG) where each node represents a transformation stage. Figure 2.3 shows a sample lineage graph of combining RDDs from two inputs. When a failure occurs at a certain stage, Spark uses the last available working point(RDD) from the lineage graph and restart all computations from that working point rather than repeating the entire process from the beginning or saving the intermediate results and replicating them across multiple nodes. This strategy of data management, fault tolerance and in-memory processing makes Spark to do computations faster than MapReduce.

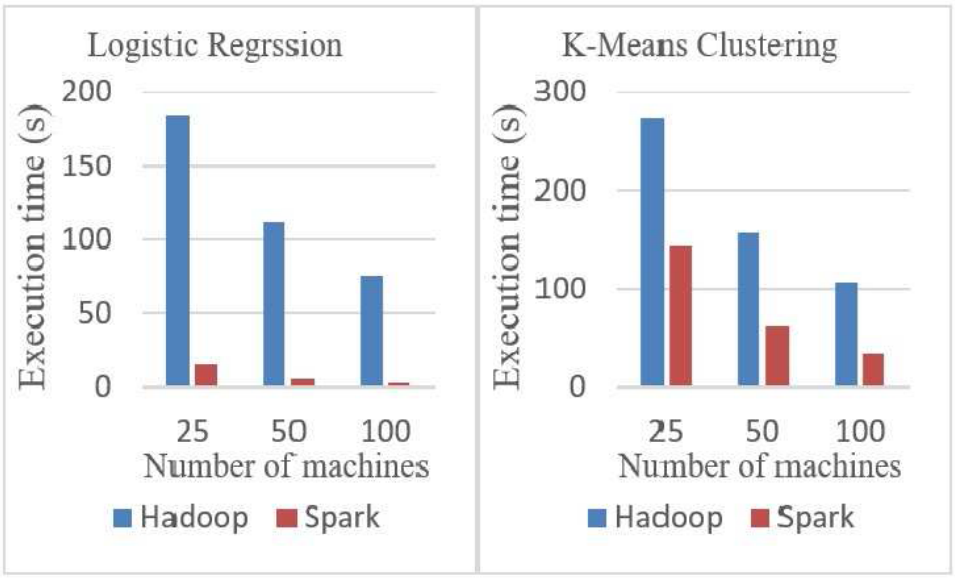


Figure 2.4: Comparison of execution time of MapReduce and Spark for Logistic Regression and K-Means Clustering

2.15 Spark Machine Learning Library - MLlib

Apache Spark [19] has Spark MLlib [47] 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.Figure 2.4 compares Hadoop MapReduce and Spark when running Logistic Regression and K-Means clustering algorithms in different number of machines for a 100GB of data [19].

Spark also provide many packages like Spark SQL, Spark Streaming and GraphX [48]. Spark SQL can query any tables from databases like Hive, Cassandra, etc. On the other hand, it can also create tables in the databases from the raw data. Spark

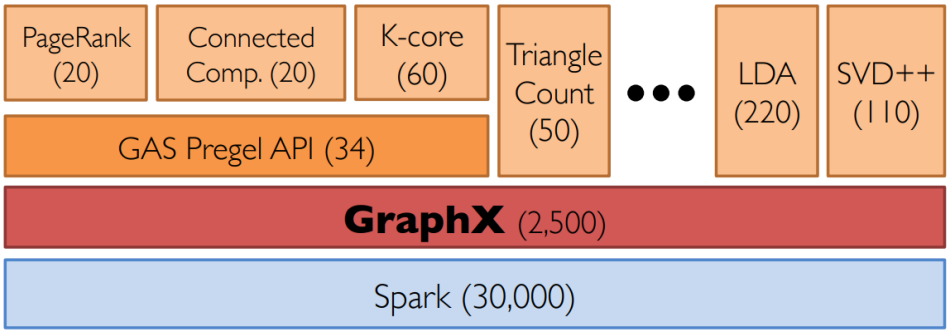


Figure 2.5: GraphX framework with its functionalities such as PageRank,LDA,Triangle COunting,Pregel,etc.

Streaming can manage a data stream from Kafka 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.

2.16 Spark Large Graph Processing - GraphX

Spark, with its efficiency in Resilient Distributed Datasets (RDDs) help wide variety of applications such as Machine Learning with MLlib library [47], Graph Analysis with GraphX library [48]. GraphX is an embedded graph processing framework built on top of Apache Spark. In general, graphs can be represented as *G=(V,E)*, where V is the set of vertices in G and E, which takes the general representation as $e_{ij} = Edge(i, j)$, is the set of edges connecting 2 vertices (i,j) in G. GraphX treats the complete graphs as an RDD. It maintains the graph RDD in the type of [VD, ED], where *VD* and *ED* are other RDDs representing vertex properties and edge properties respectively. Figure 2.5 provides the simple GraphX framework and functions it provide to support various graph operations. GraphX performs graph-specific operations as a series of distributed *map()*, *join()* and *reduce()* functions of RDDs. Besides these functions, GraphX comprise of Google's *Pregel API* [49]. GraphX uses Pregel API to perform iterative tasks like PageRank, Graph search algorithms like Depth First

Search (DFS) and Breadth First Search (BFS) and finding shortest routes in graphs like Dijkstra's algorithm. In iterative graph algorithms, vertices of the graph have to pass some messages to their neighbors. Since the graph is maintained as a single RDD in GraphX, the message passing is complicated compared to other graph libraries. The Pregel API automates this message sending and receiving module and provides a functionality to do these jobs efficiently to suit the Spark environment. GraphX also shows great speedups for iterative graph algorithms such as PageRank compared to other graph libraries such as GraphLab [50] and Giraph [51]. For iterative graph processing, GraphX provides *Pregel API* [49]. Pregel works in a message passing fashion between the graph vertices. In GraphX, Pregel has three functions: *sendMsg()* - to process and send a message to a vertex's immediate neighbors, *mergeMsg()* - to merge all messages from a vertex's immediate neighbors and *receiveMsg()* - to receive and process the merged message. Following these steps, each vertex can share and collect information with their neighbors. With this method, the information can flow from one end of the graph to another gradually. For iterative procedure, Pregel iterations are named as *SuperSteps*. In each SuperStep, each vertex executes all three above mentioned functions.

2.17 Graph Search Algorithms

2.17.1 Dijkstra's Shortest Path Algorithm

Consider a weighted graph $G = (V, E)$, where V is a finite set of vertices and E is the set of edges connecting vertices in graph G with some weight w_i is assigned to edges $e_i \in E$. Dijkstra's Shortest path algorithm helps to find a shortest distance between a source vertex u and a destination vertex v of the given graph G, where $u, v \in V$. The algorithm starts with the source vertex u and decides on which vertex (say $u + 1$) to visit next, in order to reach the destination vertex v in a shortest distance. The node u chooses a next node to visit based on which vertex is in the shortest distance from u. This method continues in all vertexes until they reach the

destination vertex v. The given algorithm process in $O(V^2)$ run time complexity. However, the complexity can be reduced to $O(ElogV)$ if the graph G is represented as an adjacency list using Heap operations [52].

Algorithm 1 Dijkstra's Shortest Path algorithm for a Graph

Require: $G = (V, E)$, source vertex u

```
    for i in range(0,V.count()) do
 2:     dist[i] = ∞
    for i in range(0,V.count()) do
 4:     visitSet[i] = false
    dist[i] = 0
 6: procedure MINDIST(dist,visitSet)
        min:= ∞
 8:     for i in range(0,V.count()) do
            if dist[i] < min and visitSet[i]=false then
10:             min := dist[i]
                index := i
12:     return index
    for i in range(0,V.count()) do
14:     u ← minDist(dist, visitSet)
        visitSet := True
16:     for j in range(0,V.count()) do
            if visitSet[j] = falseanddist[j] > (dist[i] + distance[i][j]) then
18:             dist[j] := dist[i] + distance[i][j]
```

2.17.2 Breadth First Search algorithm

Consider a weighted graph $G = (V, E)$, where V is a finite set of vertices and E is the set of edges connecting vertices in graph G. Algorithm 2 gives an overview of a simple Breadth First Search algorithm for a graph G. Breadth First Search algorithm operates in $First - In - First - Out(FIFO)$ fashion. A queue is created for the graph G to track the graph traversal and an arbitrary vertex $u \in V$ is selected to begin iterations. In each iteration, a vertex s from the queue is popped out and all their neighbors except visited from the graph G are selected and pushed into the queue. The process continues until all nodes in the graph G are visited and the queue

becomes empty. Given a graph $G = (V, E)$, the BFS algorithm runs in $O(V + E)$ complexity.

Algorithm 2 Breadth First Search algorithm for a Graph

Require: $G = (V, E)$, start vertex u

```
    q := Queue()
 2: q.push(u)
    for i in range(0,V.count()) do
 4:     visited[i] = false
    while !q.isEmpty() do
 6:     s := q.pop()
        s_n ← G.get_neighbors(s)
 8:     for v in s_n do
            if !visited[v] then
10:             q.push(v)
```

2.17.3 Depth First Search traversal

Consider a weighted graph $G = (V, E)$, where V is a finite set of vertices and E is the set of edges connecting vertices in graph G. Depth First Search algorithm operates in $Last - In - First - Out(LIFO)$ fashion. A stack is created for the graph G and an arbitrary vertex $u \in V$ is selected to begin the process. Once the vertex is selected, they are added to the stack. To iterate over the process, a last vertex from the stack is pulled and their neighbors from the graph G are selected and pushed into the stack. The process continues until all nodes in the graph G are visited and the stack goes empty.

Algorithm 3 Depth First Search algorithm for a Graph

Require: $G = (V, E)$, start vertex u, destination vertex v

```
    Mark all vertexes that belong to same attributes as u or v as visited and set other
    vertexes as not visited
 2: s := Stack()
    s.push(u)
 4: s.setVisited
    while !s.isEmpty() do
 6:     c ← s.get_last_node()
        c_n ← c.get_first_unvisited_neighbor()
 8:     s.add(c_n)
        if c_n = null then
10:         s.remove(c)
            continue
12:     if c_n = v then
            break
14:     elsec_n.setVisited
```

CHAPTER 3: ACTION RULES MINING ALGORITHMS

Huge amount of work has been done previously on topics of Action Rules, Cloud Computing and Distributed rule mining In this chapter, we give brief history of other works related to our research on the above mentioned topics.

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 [53] [54] and object-based approach [28] [55] [56].

3.1 Rule-Based Action Rules Extraction Algorithms

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 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:

3.1.1 LERS

All rule-based approaches in action rule mining use Learning from Examples using Rough Sets (LERS) [57] 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. Algorithm 4 shows an overview of LERS algorithm. 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 $a \in A$ and X_v be the objects supporting an attribute value v. Thus, for the decision table in Table 2.1, 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 2.1, the classification rules from LERS would be:

$$a_0 \rightarrow d_2 \ , \ a_1 \rightarrow d_1 \ , \ a_2 \rightarrow d_1 \ b_2 \rightarrow d_1 \ , \ c_1 \rightarrow d_2$$

Algorithm 4 LERS

Require: a_s and d_s, where a_s and d_s are dictionaries to store all distinct attribute values and their corresponding object ids from the decision table S

```
1:  procedure PROCEDURE
2:      a_s ← {values from all attributes} - {decision attribute values}
3:      d_s ← {decision attribute values}
4:      fixedValues ← a_s
5:      while a_s ≠ {} do
6:          for < key, value > in a_s do
7:              if value ⊆ one of the values of d_s then
8:                  certainRules ← (key, decisionValue)
9:              else
10:                 possibleRules ← (key, value)
11:             delete key from a_s
12:         for (k_1, v_1) in possibleRules do
13:             for ( do(k_2, v_2) in fixedValues)
14:                 if k_2 ⊆ k_1 then
15:                     Continue
16:                 else
17:                     k_3 ← (k_2, k_1)
18:                     v_3 ← Set of object from S supporting k_3
19:                     a_s+ = (k_3, v_3)
```

3.1.2 System DEAR

Authors Tsay, et.al. [53] 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 to LERS [57] 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.

3.1.3 System ARAS

Ras, et. al [54] 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 [57] algorithm to generate classification rules. Consider the classification rules for the decision table in Table 3.1 generated by LERS and consider that the user prefers to change the decision from d_1 to d_2. 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 Table 2.1, ARAS generates only one action rule schema AR_s as given in Equation 3.1

$$AR_s = (\mathbf{A}, \rightarrow a_0) \longrightarrow (\mathbf{D}, d_1 \rightarrow d_2) \tag{3.1}$$

For the action rule schema $\mathrm{AR_s}$, let $\mathrm{V_{st}}$ be the stable attributes, $\mathrm{V_{fl}}$ 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 $X_{\mathrm{AR}} \not\subseteq X_{\mathrm{ARs}}$. Some of the Action Rules from system ARAS and DEAR 3 for the decision table Table 2.1 are given in Equation 3.2, Equation 3.3 and Equation 3.4.

$$(\mathbf{A}, a_1 \rightarrow a_0) \Longrightarrow (\mathbf{D}, d_1 \rightarrow d_2) \tag{3.2}$$

$$(\mathbf{A}, a_2 \rightarrow a_0) \Longrightarrow (\mathbf{D}, d_1 \rightarrow d_2) \tag{3.3}$$

$$(\mathbf{A}, a_1 \rightarrow a_0) \wedge (\mathbf{C}, c_2) \Longrightarrow (\mathbf{D}, d_1 \rightarrow d_2) \tag{3.4}$$

Thus, ARAS system 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.

3.1.4 Object-Based Action Rules Extraction Approaches

Object-based approaches as proposed in [28] [55] [56], 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:

3.1.5 Association Action Rules

Authors Ras et. al [28] propose a method of extracting new form of Action Rules from the given information system S using Apriori like algorithm [26] 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 decision table S given in Table 2.1, Association Action Rules algorithm generates following itemset pairs:

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

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.

3.1.6 Object Driven Action Rules

Authors A. Hajja et. al [55] propose a new dimension of action rule as *object-driven Action Rules*. This system works on object-driven information system. The Decision table S changes to Object-driven decision table S_o when some instances in the decision table belong to an object 'm_i'. This way, the instances can group into 'm' clusters where 'm' is the number of objects in the decision table. In addition, this system introduces the notion of temporal constraint into the decision table. 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 [28] to extract Action Rules from this new information system.

Table 3.1: Sample Decision Table 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 3.1 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 [55] can cause over-fitting problems by individualiz-

ing each objects 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 in [56] 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 4.1 shows the effect of clustering in terms of number of Action Rules and their total support from [56].

Table 3.2: 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

3.1.7 LISp-Miner

Rauch and Šimůnek [58] 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

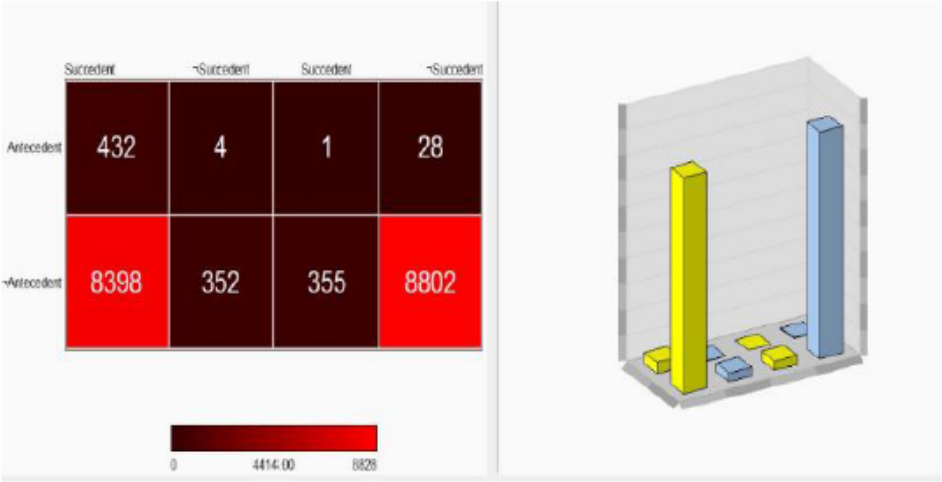


Figure 3.1: Confusion matrix and histogram of an action rule for a BMI dataset in LISp-Miner

advanced version of association rules. Act4ft-Miner produces G-Action Rule R in the form as represented in Equation 3.5

$$\phi_{\mathrm{St}} \wedge \Phi_{\mathrm{CHg}} \approx * \psi_{\mathrm{St}} \wedge \Psi_{\mathrm{CHg}} \tag{3.5}$$

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 output some visualizations in the form of confusion matrix and its corresponding histogram for each extracted action rule. Figure 3.1. shows a sample visualization of a confusion matrix and corresponding histogram for an action rule.

3.2 Discovery of Action Rules of Lowest Cost

Action Rule patterns are interesting, if the extracted rules are diverse. The patterns are diverse if its elements differ significantly from each other. The more diverse the extracted patterns are, the more interesting they are. Action Rules of low cost are considered patterns of high interest. The cost is a subjective measure, in a sense that domain knowledge from the user or experts in the field is necessary in order to determine the costs associated with taking the actions.Action Rules costs the user or company in some form of money or resources like energy, power, human resources or even a moral value to make the recommended changes to achieve the desired action or goal. For example, lowering the interest percent rate for a customer is a monetary cost for the bank; while, changing the marital status from 'married' to 'divorced' has a moral cost, in addition to any monetary costs which may be incurred in the process. However, most of the Action Rules extraction algorithms does not guarantee the cost economic recommendations to the user or company. To address this problem, Ras and Tzacheva [4] introduced the notion of a cost and feasibility of an Action Rule. They suggest a heuristic strategy for creating new Action Rules, where objects supporting the new Action Rule also support the initial Action Rule but the cost of reclassifying them is lower or even much lower for the new rule. In this way, the rules constructed are of more interest to the users and in the context of a business action plan.

Tzacheva, et. al proposed a new method to extract low cost Action Rules and their generalizations taking into account of correlations between individual atomic action sets [59]. An action set is called *n-pair* set, if it comprises of n action terms in it. We can iteratively create *2-pair*, *3-pair*,.....,*n-pair* sets. From a list of action rules, we extract all atomic action sets. Next, we build a *Correlation Matrix* which shows the most frequent pairs of atomic action sets within the list of action rules. Figure 3.2 shows a sample 1-pair Correlation matrix. An atomic action set pair is said to be $frequent$ if it satisfies the minimum frequency threshold ϑ specified by the user. We

$e_? \rightarrow e_2$							
$e_1 \rightarrow e_2$	0						
$e_2 \rightarrow e_3$	0	0					
$f_? \rightarrow f_2$	1	0	0				
$f_3 \rightarrow f_2$	0	0	1	0			
$g_? \rightarrow g_1$	0	0	1	1	2		
$g_3 \rightarrow g_1$	1	1	0	2	0	0	
	$e_? \rightarrow e_2$	$e_1 \rightarrow e_2$	$e_2 \rightarrow e_3$	$f_? \rightarrow f_2$	$f_3 \rightarrow f_2$	$g_? \rightarrow g_1$	$g_3 \rightarrow g_1$

Figure 3.2: Example 1-pair Correlation matrix for finding low cost Action Rules from the decisiont table S

scan the correlation matrix, and we *mark* all *1-pair sets* which are found to be *frequent*. Then a *2-pair* correlation matrix is built by combining the marked from the *1-pair* correlation matrix. The process is repeated with *3-pair, 4-pair, ... , n-pair* correlation matrix, until no more action set pairs are marked.

CHAPTER 4: DISTRIBUTED ACTION RULES MINING

In this chapter, we describe all of our proposed methods to extract Action Rules, which scale for large data processing, using distributed computing frameworks such as Hadoop MapReduce [18] and Spark [19]. We also discuss briefly the challenges involved in performing Action Rule extraction in a distributed environments. For all the Action Rule extraction algorithms, we give three inputs: *data*, *attributes* and *parameters*, where the *data* is a given data without any attribute or column names, *attributes* is an input file with all attribute names (A) and *parameters* is another input file with basic parameters such as a set of stable attributes (A_{st}), where $A_{st} \subseteq A$, decision attribute (A_d) and conditions that every Action Rule has to satisfy such as minimum support (m_{s}), minimum confidence (m_{c}) including *decision from* (d_from) and *decision to* (d_to) values.

Figure 4.1 gives an overview of our proposed methods for extracting Action Rules using distributed computing frameworks. We take two of the existing Action Rules extraction methods: ARoGS(the fastest method) and Association Action Rules(the slowest method) as a motivation for all of our methods. We propose two varieties of ARoGS method: SARGS(using Apache Spark) and MR-Random Forest(using Hadoop MapReduce), and two varieties of Association Action Rules method: Vertical Data Distribution and Semantic Data Distribution. Apart from from proposing these methods, we also give plugin modules such as Class Data Distribution and Data Granules Approach to partition/divide the given massive data into chunks. We are giving this data partition methods as plugins because of the resulting data chunks can be used by any distributed action mining algorithms.

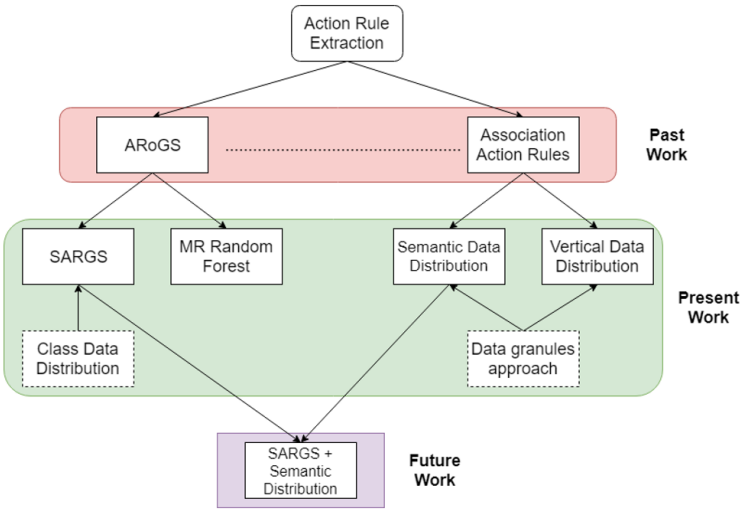


Figure 4.1: Overview of proposed methods to extract actionable patterns in a distributed setup

4.1 Distributed Rule Mining Algorithms

Rule based Machine Learning intends to circumscribe methods that identifies, learns or evolves *rules* to store, and manipulate knowledge [60]. 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. Due to racing volume of big data, many researches in the history have adopted distributed processing frameworks such as Hadoop MapReduce [18] and Apache Spark [19] and help generating classification rules [61] [62] [63] for classification or association rules [64] [65] [66] to get associativity between items in a short time for a large volume of data.

4.2 Distributed Classification Rules Mining

Classification is a process of allotting data objects to one of the several categories or classes [67]. In other words, classification is a supervised machine learning process that learns a target function $\boldsymbol{f}$ that maps attribute set x to one of the labels y. For

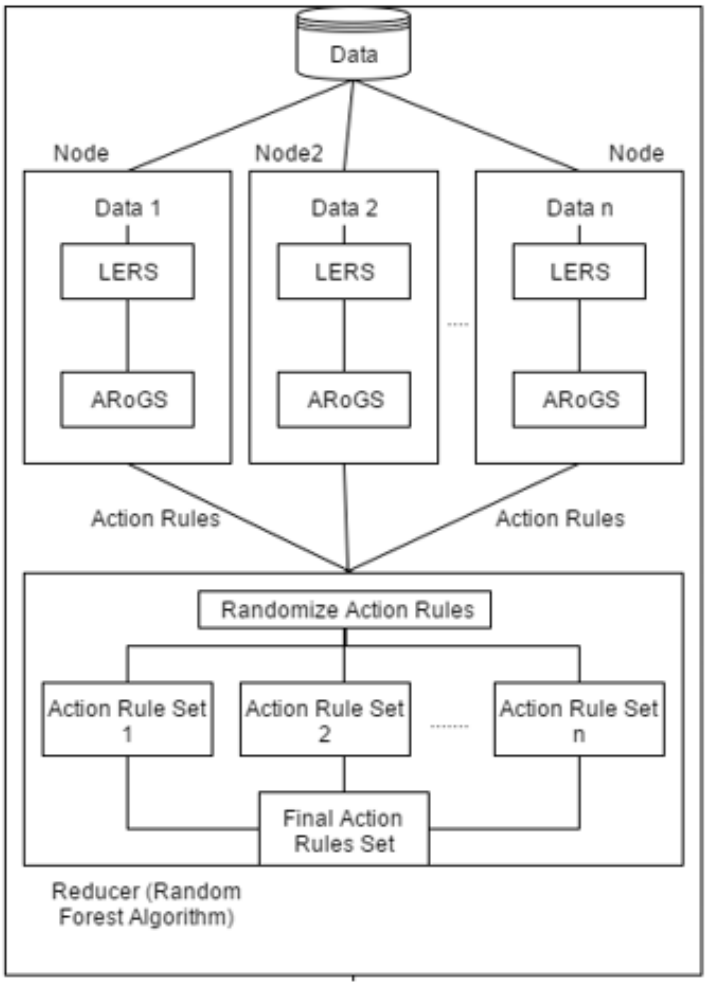


Figure 4.2: Overview of MR-Random Forest Algorithm to extract Action Rules using LERS and ARAS and methods

classification, 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, ... , 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 [68]. Direct methods 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 proposed a distributed computation of combination of decision tree algorithm C4.5 [61] and ensemble learning method called *Bagging* [69] 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 braches giving n different nodes, where n is a number of distinct values in A_s. C4.5 algorithm [60] 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, ... , 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 x 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 et. al. proposed a MapReduce implementation of traditional decision tree algorithm C4.5 [62]. Considering the communication cost in the MapReduce model, they 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 proposed a direct method of extracting classification rules in MapReduce framework [63]. 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 4.1 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 4.1: Accuracy comparison on RuleMR with other Classification 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 [63] and faster results.

4.2.0.1 Distributed Association Rules 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 [26]. 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:

Lin proposed a MapReduce [18] based approach to extract association rules [64]. 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.

Qin et. al in [65] modified the above discussed Apriori model and proposed Yet Another Frequent Itemset mining(YAFIM) in Spark framework [19]. 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 [26].

Rathee et. al proposed *Reduced-Apriori* or *R-Apriori* [66] to speed up the algorithms proposed in [64] and [65] using Spark [19]. 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 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 multiple nodes gives the final count of the new frequent itemset pair. Fig. 4.3 shows the speed comparison of methods proposed in [64] [65] and [66] 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.

4.2.1 Granular Computing in Data Mining

Although Granular computing was originally intended by Zadeh [40] purely to represent human cognition, the idea of the topic has been adopted in multiple problems like decision trees [70], divide and conquer [71], set theory, data reduction or compression, and discretization [72]. One of the applications of information granules are

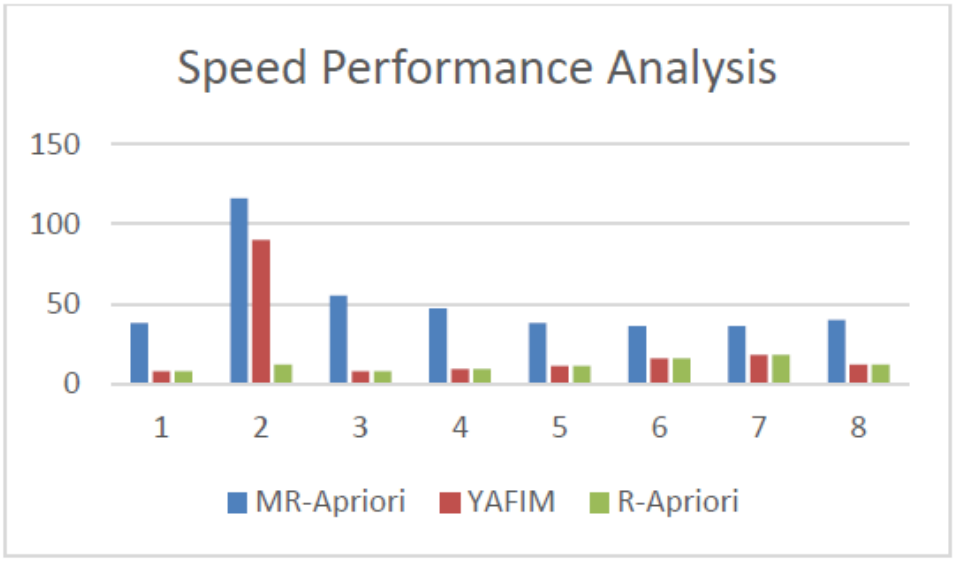


Figure 4.3: Performance Analysis in iterations of systems MR-Apriori, YAFIM and R-Apriori

finding optimal solution that satisfies certain imprecise human assigned conditions [71]. Granular computing has been used in certain image processing applications, by grouping some pixels into semantically meaningful clusters or granules[73]. Recently, Liu, et.al [74] used information granules for time series models.

4.3 Simple Data Distribution methods

4.3.1 MR-Random Forest Algorithm - LERS and ARAS based

Algorithm 5 AR

Require: certainRules, d_{from} and d_{to}, where certainRules are provided by Algorithm LERS

1: **procedure** PROCEDURE
2: **for** (k, v) in certainRules **do**
3: **if** $value == d_to$ **then**
4: $actions \leftarrow \{\}$
5: **for** $a \in key$ **do**
6: $A \leftarrow attributeName(a)$
7: $actions+ = (A, \rightarrow a)$
8: **Output** actions

Algorithm 6 ARAS

Require: actions, d_{from} and d_{to}, where *actions* is a list of actions from Algorithm **AR**

1: **procedure** PROCEDURE
2: $s_V \leftarrow$ list of stable attribute values in *actions*
3: $m_V \leftarrow$ {missing flexible attribute values in actions}
4: $actionSupport \leftarrow$ {set of objects in S supporting $s_V \cap d_from$}
5: **for** $value \in m_V$ **do**
6: $newValues \leftarrow value + s_V$
7: $newSupport \leftarrow$ set of objects in S supporting $newValues$
8: **if** $newSupport \subseteq actionSupport$ **then**
9: $actions+ = value$
10: **Output** actions as Action Rule

This is our first step on using cloud based techniques to extract Action Rules. In this method, we select the ARAS algorithm [54] in association with LERS to get Action Rules in a Hadoop MapReduce environment. Figure 4.2 gives an overview of this proposed method [75]. Our MapReduce job consists of Map and Reduce parts. The algorithms are designed to implement the Map part. Hadoop splits the data and give splits of data to several Map parts. All mappers start executing Algorithms 4, 5 and 6, where the algorithm *LERS* outputs classification rules, algorithm *AR* processes classification rules and give inputs to Algorithm *ARAS*. Algorithm *ARAS* inturn outputs Action Rules. The resulting action rules from all the Maps are combined in such a way that the action rule acts as a key and the support and confidence from all the maps acts as iterator list of values. The combined action rules are given to the Reduce part which implements Random Forest algorithm. This algorithm check if an action rule is resulting from 50% of the Maps. If so it averages all supports and confidences and check them against the minimum support and confidence. Algorithm 7 gives the overview of how the Reduce part works.

Algorithm 7 MR-Random Forest Reduce algorithm

Require: **key** - action rule and **values** - list of support and confidence for action rule

1: **procedure** PROCEDURE
2: **if** $|values| \geq n/2$ **then**
3: supp $\leftarrow$ Mean of all supports in *values*
4: conf $leftarrow$ Mean of all confidence values in *values*
5: **if** supp $\geq$ minimum support and conf $\geq$ minimum confidence **then**
6: **Output** Key with *supp* and *conf*

4.3.2 MR Apriori - Association Action Rules based

In this section, we describe our adaptation of the Association Action Rules method from section 3.1.5. in Hadoop MapReduce [18] Algorithm 8 gives an overview of Association Action Rules algorithm in a distributed environment. The frequent action set generation is divided in two steps- merge and pruning. In merging step, merge previous two frequent action sets into a new action set and in the pruning step, discard the newly formed action set if it does not contain the decision action. From each frequent action set, the association action rules are formed. Algorithm **AAR** generates frequent action sets and association action rules from the action sets.

Algorithm 8 AAR

1: **procedure** PROCEDURE
2: primaryActionSets $\leftarrow$ {all possible atomic action terms from decision table S}
3: tempActionSets $\leftarrow$ primaryActionSets
4: $newActionSets \leftarrow \emptyset$
5: **while** $tempActionSets \neq \emptyset$ **do**
6: **for** $set1 \in tempActionSets$ **do**
7: **for** $set2 \in primaryActionSets$ **do**
8: **if** ($decisionAtomicAction \subseteq set1$ or $decisionAtomicAction \subseteq set2$ and $set2_{attributes} \not\subseteq set1_{attributes}$) **then**
9: $newActionSets+ = set1 \cup set2$
10: **Output** *newActionSets*
11: $tempActionSets \leftarrow newActionSets$

Figure 4.4 gives an overview of this proposed method [75]. In Hadoop MapReduce framework, the Association Action Rules method runs on mappers to produce an

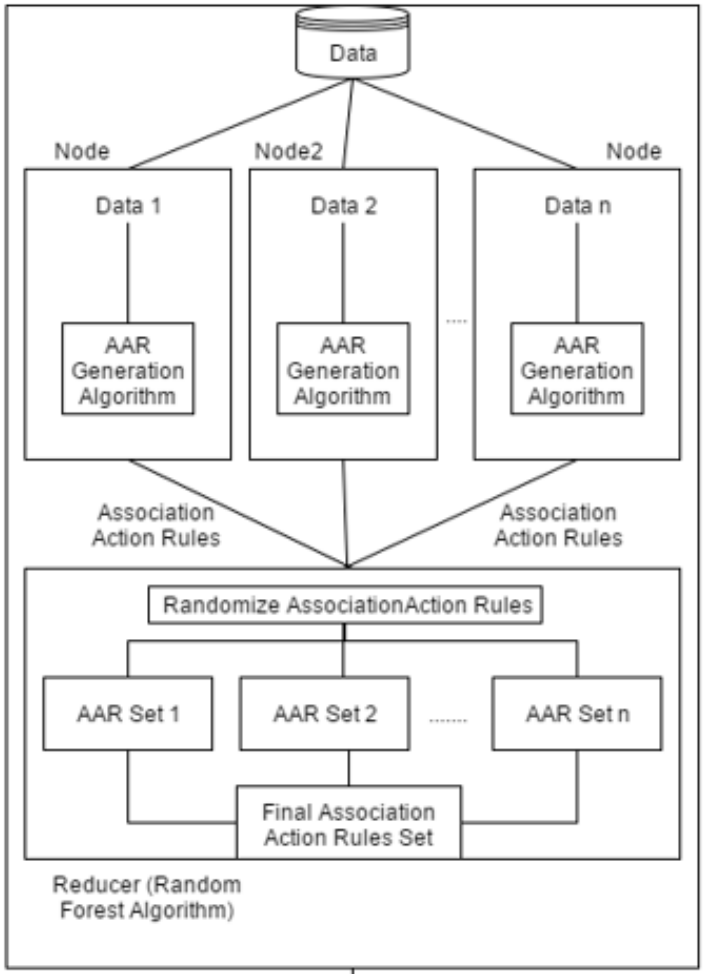


Figure 4.4: Overview of MR-Apriori Algorithm to extract Action Rules using Association Action Rules method

Association type of Action Rules (AAR). Hadoop splits the data and give splits of data to several Map parts. The resulting action rules from all the Maps are combined in such a way that the action rule acts as a key and the support and confidence from all the maps acts as iterator list of values. The combined action rules are given to the Reduce part which implements Random Forest algorithm. This algorithm check if an action rule is resulting from 50% of the Maps. If so it averages all supports and confidences and check them against the minimum support and confidence. Algorithm Reduce gives the overview of how the Reduce part works.

4.3.3 SARGS - Specific Action Rule discovery based on Grabbing Strategy

In our next work [76], we proposed an alternate version of *ARAS* method: *Specific Action Rule discovery based on Grabbing Strategy* (SARGS) using the Spark framework [19]. Action Rules from the *ARAS* algorithm only one specific atomic action term. The left side of other action terms are empty. These Action Rules can give only a limited knowledge to the user and leave clueless due to the lack of specific action terms. We proposed *SARGS* method to fill all missing values in the action rule schema produced by *LERS* algorithm in an efficient time using Spark framework.

Algorithm 9 SARGS

Require: actions of type list(actions) and d_{FROM} values

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

Algorithm 9 gives the modified version of ARAS module that the SARGS algorithm uses to extract all complete Action Rules. This algorithm extracts all missing values from the conditional (left) part of the given Action Rule schema. The algorithm then get cartesian product of all missing values (except the values of same attribute) and fills in the action rule.

In Spark, reading each file: attributes, parameters and data creates three different RDDs. Spark has options to broadcast certain values among all worker nodes. We broadcast values that we read from attributes and parameters file, so that all nodes can access them. The input data file gets partitioned depending on its size.

Figure 4.5 gives an overview of SARGS execution. Once we distribute the given

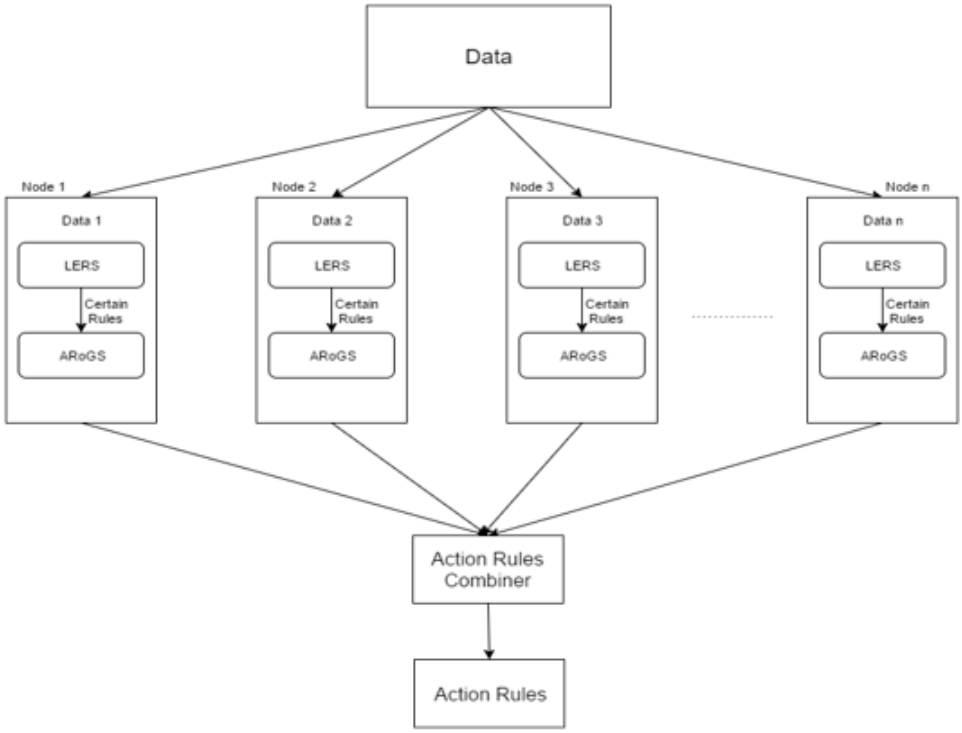


Figure 4.5: Distributed Actionable Pattern Mining using Spark framework

data using the above mentioned method, Spark splits the data to d partitions. Algorithms *LERS* and *SARGS* run on each of RDDs, created by reading the data file, using MapPartition function, which is used to perform computations on each and every partition of data. Each partition then results in their own set of Action Rules with support and confidence.

All Action Rules from the MapPartition function are sorted by the attribute name and returned as (Key, Value) pairs. We choose 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 $fs >= m_s$ and $fc >= m_c$.

4.4 Advanced Data Distribution Methods

4.4.1 Class Attribute Sampling

In the MR-Random Forest algorithm [75] or in SARGS algorithm [76], Hadoop or Spark manages data distribution over the nodes in a cluster and all algorithms – ARAS [54] and Association Action Rules [28], run on top of such partitions using MapReduce or Spark frameworks. When dsitributed computing frameworks manage 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. Thus we introduce a new data distribution module into our SARGS method, which executes before the execution of SARGS algorithm. This method employs a class attribute value strategy to distribute the data, where the class attribute values and their relative percentages are matched in each partition . It can be applied to any Action Rule discovery algorithms, including ActionRules[27], system DEAR [53], ARAS [28]. We propose a method similar to Stratified Sampling [77] for data distribution to all partitions. We split the given data into groups where each group consists of records matching single decision value. Also, we maintain the proportion constraint $P_g \simeq P_S$, where P_g is the proportion of records in a partition g with decision attribute value d_i and P_S is the proportion of records in the given information system S with decision attribute value d_i. By this way, each partition contains same proportion of data which is equal to the original dataset.By this way, each partition contains same proportion of data which is equal to the original dataset. Figure 4.6 shows an example data partition for the decision table S given in Table 2.1.

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

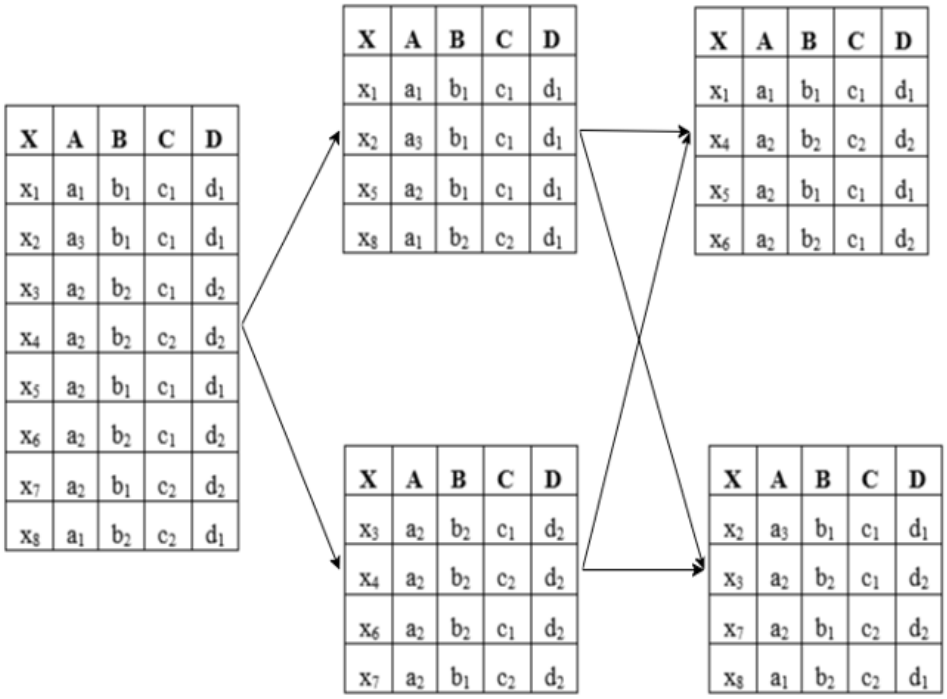


Figure 4.6: Data Distribution approach similar to Stratified Sampling

LERS (Algorithm 4)and *SARGS* (Algorithm 9) execute on each of these partitions and final Action Rules are grouped together.

4.4.2 Vertical Data Partitioning

In this work[78], we propose a novel approach for extracting Action Rules by splitting the data vertically, in contrast with the classical horizontal split, which is performed by parallel processing systems. This vertical split method can be applied to the Association Action Rules discovery method described by Ras et. al [28]. Association Action Rules is an exhaustive A-Priori like method, which extracts all possible action rules by taking all combinations of ActionTerms through iterative nature. For that reason, Association Action Rules is the most complex, and the most computationally expensive out of all Action Rules extraction algorithms. The method does not scale very well with Big Data and with High Dimensional data, and suffers from performance time prospective. Therefore, this algorithm requires attention, and needs to be adapted for scalable and faster computation. In this work, we propose the Vertical

Data Split method, which allows for much faster computational time for Association Action Rules extraction, as well as it it makes it possible to run the extraction in a Cloud Cluster environment in parallel.

On the positive side, this Association Action Rules algorithm extracts all possible ActionRules, while other algorithms, have a chance to lose some valuable rules. For that reason, Association Action Rules execution takes much longer than other Action Rules extraction algorithms. Association Action Rules algorithm is similar to Association Rules [79]extraction algorithm. Association Rules find patterns that occur most frequently together in the given data. The most popular algorithm for extracting Association Rules is the Apriori algorithm [66]. Apriori algorithm starts with 2 element pattern and continues n iterations until it finds n element patterns, where n is the number of attribute in the given data. Sample Association rule, which means that when a pattern $a_1 \cap b_2$ occur together in the data, pattern $c_1 \cap d_2$ also occurs in the same data, are given below in Equation 4.1

$$(a, a_1) \cap (b, b_2) \longrightarrow (c, c_1) \cap (d, d_2) \tag{4.1}$$

Action Rules also have relation to the Association Rules. When an actionable pattern $t_1 \cap t_2 \cap .... \cap t_n$ occurs, where t_i represents an *atomic action*, the actionable pattern based on the decision attribute d_i also occurs with minimum support(m_s) and confidence(m_c). Association Action Rules starts extracting Action Rules with a pattern which comprises an atomic action on the left side of the rule and decision action on the right side. The algorithm continues for maximum of *n-1* iterations, where n is number of attributes in the data and gives all actionable patterns in the data. We propose a method provides very broad recommendations and works comparatively faster than our previous approaches: MR-Random Forest algorithm [75] and SARGS algorithm [76]. We propose this method that suits data that has large attribute space. But the method works better with small data also. In this

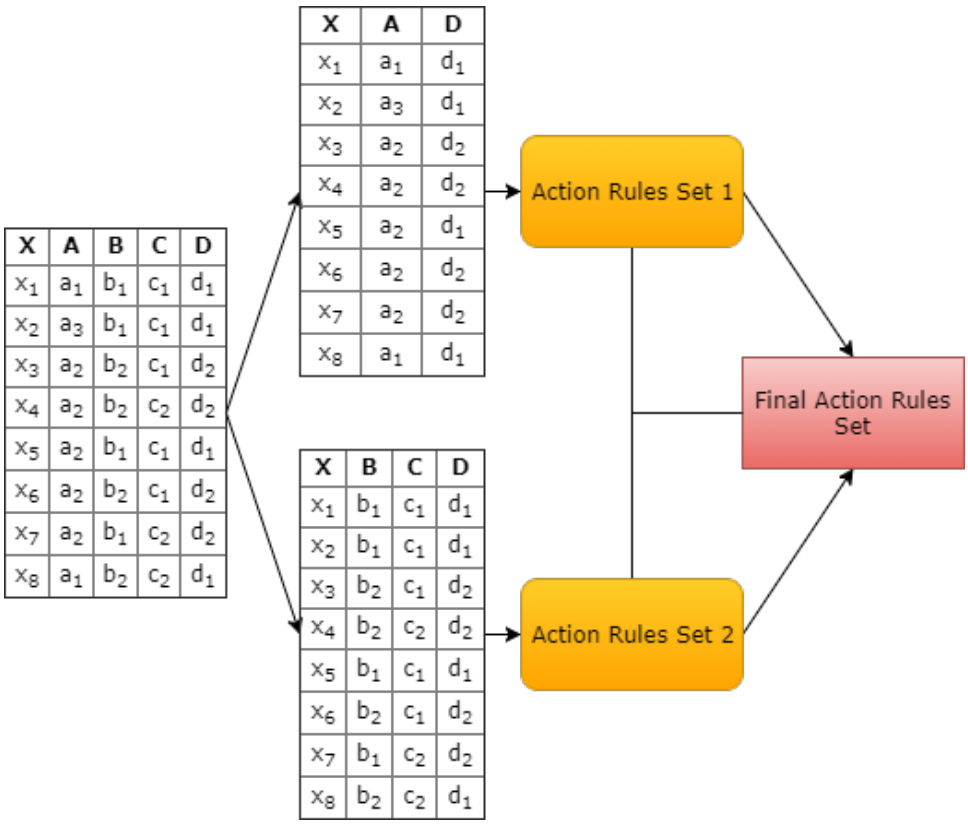


Figure 4.7: Example Vertical Distribution for Decision Table in Table 1

method, we split the data vertically into 2 or more partitions and with each partition the data can be split horizontally by the default settings of the Spark framework. Figure 4.7 presents an example vertical data partitioning with the sample Decision table given in Table 2.1. Our algorithm runs separately on each partition, does transformations like *map()*, *flatmap()* functions and combine results with *join()* and *groupBy()* operations. Algorithm 10 gives our new algorithm to extract all possible Action Rules from the data in a parallel fashion.

Our algorithm gets the pre-processed data (r_id, r_values) as input, where r_id is the *row id* and r_values is a list of values for each record in the data. The algorithm also takes **decision from** (d_{FROM}) and **decision to** (d_{TO}) values as parameters. *Step 2* of the algorithm gets all distinct attribute values and their corresponding data row indexes. This step involves a Map phase and a groupByKey phase of the Spark frameworks. We collect the data row indexes to find the Support and Confidence of

Algorithm 10 ActionRulesExtract

Require: **data** of type (r_{id}, r_{values})) and d_{FROM}, d_{TO} values

procedure MyProcedure

$d_A := (s \in r | r \in data | (s, r_{id})).groupByKey()$

$c_{OLD} \leftarrow d_A$

$i \leftarrow 2$

parallel:

while i $\neq$ n **do**

$c \leftarrow data.flatmap(r => (comb(r_{values}, i)), r_{id})$

$c_{NEW} \leftarrow c.groupByKey()$

$c_{VALID} \leftarrow c_{NEW}.filter()$

$c_{FROM} \leftarrow c_{VALID}.filter(d_{FROM})$

$c_{TO} \leftarrow c_{VALID}.filter(d_{TO})$

if $c_{FROM} = \emptyset$ or $c_{TO} = \emptyset$ **then break**

$atomic \leftarrow c_{FROM}.jon(c_{TO}).filter()$

$action_{supp} \leftarrow (r \in atomic \mid findSupp(r)).filter()$

if $action_{supp} = \emptyset$ **then break**

$atomic_{FROM} \leftarrow atomic.filter()$

$atomic_{TO} \leftarrow atomic.filter()$

$a_{FROM} \leftarrow atomic_{FROM}.join(c_{OLD})$

$a_{TO} \leftarrow atomic_{TO}.join(c_{OLD})$

$action_{conf} \leftarrow a_{FROM}.join(a_{TO})$

$actions \leftarrow action_{supp}.join(action_{conf})$

collect $actions$

$c_{OLD} := c_{NEW}$

$i := i + 1$

Action Rules.

Finding Support and Confidence is an iterative procedure. It takes **O(nd)** times to collect Support and Confidence of all Action Rules, where **n** is the number of Action Rules and **d** is the number of data records. To reduce this time complexity, we store a set of data row indexes of each attribute value in a Spark RDD. In *Step 3*, we assign the distinct attribute values to old combinations (c_{OLD}) RDD to start the iterative procedure. Thus the d_A RDD acts as a seed for all following transformations. The algorithm runs maximum of *(n)* iterations, where n is the number of attributes in the data. During the i^{th} iteration, the algorithm extracts Action Rules with *i-1* action set pairs on the left hand side of the rule. *Step 8* uses *flatMap()* transformation on the data to collect all possible i combinations from a data record. We sort the combination of attribute values since they act as key for upcoming *join()* and *groupBy()* operations. We also attach a row id r_{id} to all combinations to get the support (which data records contains a particular pattern) of each combination with the use of Spark's *groupByKey()* method in *Step 9*. We do sequential filtering in following steps. In *Step 10*, we filter out combinations for which the indexes count is less than the given support threshold *supp*. From the filtered combinations, we get combinations (c_{FROM} and c_{TO}) that has decision from d_{FROM} value and decision to d_{TO} values in *Step 11* and *Step 12* respectively. In *Step 14*, we join c_{FROM} and c_{TO} based on attribute names, filter outd_{FROM} and d_{TO} values since we know the decision action, which is not required in finding confidence of Action Rules. This results in Action Rules of the form $(attributes, (fromValues, fromIndexes), (toValues, toIndexes))$. We then calculate actual support of the resultant Action Rules and filter out rules that has support atleast the given support threshold *supp* in *Step 15* and reformat it to the form given in Equation 2. From *Step 14*, we have $|Y_1.Z_1|$ and $|Y_2.Z_2|$ or in other words numerator of the Confidence formula. To find the denominator $|Y_1|$ and $|Y_2|$, we again from values indexes and to values indexes in *Step 17* and *Step 18*

respectively. We perform *join()* operation with *Old combinations* and assign values to a_{FROM} and a_{TO} in *Step 19* and *Step 20* respectively. Subsequently we perform division operation of the Confidence formula in the same steps. In *Step 21*, we join a_{FROM} and a_{TO} and perform multiplication operation on the Confidence formula and reformat it the form given in Equation 2. We now join Action Rules with Support from *Step 15* and Action Rules with Confidence from *Step 21* to get final set of Action Rules. In *Step 23*, we assign new combination to the old combinations and pass the same to the next iteration.

4.4.2.1 Vertical Data Distribution using Information Granules

The basic advantage of information granularity is that we can break bigger problems into fine grained granules. Since our problem is with distribution of data, we incorporate information granules to our method[80]. Algorithm 11 gives a brief description about the process we use to measure overlaps between 2 granules, and sub-granules within each granule.

By granules, we mean a finite set of attributes from the attribute set A from the information system. For our initial experiments, we examine all combinations of 2 granules from the information system and minimize the correlations of granules given by Equation 4.2, where $C(G)$ represents correlation of a sub-granule with sub-granules of the other granule and m,n represents number of combinations of values of granules 1 and 2 respectively.

$$\frac{\sum_{i=1}^{m} C(G_i) + \sum_{j=1}^{n} C(G_j)}{2} \tag{4.2}$$

We now give an example on our optimization process for the given Information System $\mathbb{T}$ in Table 2.1. Since we are handling data partitioning, we are taking attribute types(*Stable*,*Flexible* and *Decision* attributes) into consideration. Given an information system $\mathbb{T}$, we run our optimization(*minimizing Equation 4.2*) on all gran-

Algorithm 11 Granule Based Data Distribution

Require: partitions, dataSplit1, dataSplit2
$split1Sum \leftarrow 0.0$
for data1 in dataSplit1 **do**
  $subpartitions \leftarrow[\ ]$
  $subpartitionsCount \leftarrow 0$
  for data2 in dataSplit2 **do**
    $commonLines \leftarrow data1.lines \cap data2.lines$
    if $commonLines \neq \emptyset$ **then**
      subpartitions.addAll(commonLines)
      $subpartitionsCount+ = 1$
      if $|subpartitions| == |data1.lines|$ **then**
        $split1Sum+ = 1/subpartitionsCount$
        break
$split2Sum \leftarrow 0.0$
for data2 in dataSplit2 **do**
  $subpartitions \leftarrow[\ ]$
  $subpartitionsCount \leftarrow 0$
  for data1 in dataSplit1 **do**
    $commonLines \leftarrow data1.lines \cap data2.lines$
    if $commonLines \neq \emptyset$ **then**
      subpartitions.addAll(commonLines)
      $subpartitionsCount+ = 1$
      if $|subpartitions| == |data2.lines|$ **then**
        $split2Sum+ = 1/subpartitionsCount$
        break
$split1Avg = split1Sum/|dataSplit1|$
$split2Avg = split2Sum/|dataSplit2|$
return $split1Avg - split2Avg$

Table 4.2: Sub-granules of granules: A,B and C,D

A,B	**C,D**
Y, N - $\{x_1\}$	N, D_1 - $\{x_1, x_5, x_8\}$
Y, H - $\{x_2, x_3\}$	Y, D_2 - $\{x_2, x_6, x_7\}$
N, N - $\{x_4, x_6\}$	Y, D_1 - $\{x_3\}$
N, H - $\{x_5, x_7, x_8\}$	N, D_2 - $\{x_4\}$

ules: ($\{A,B\}$ and $\{C,D\}$,$\{A,C\}$ and $\{B,D\}$ ...). Example of such combinations and sub-granules are given in Table 4.2. In the given example, the number of sub-granules, $m, n = 4$. We measure the correlation of each sub-granule($C(G_i)$,$C(G_j)$) by checking the overlap count of the sub-granule with sub-granules of other granule. For example, $C(G_{Y,H}) = \frac{1}{2}$, since Y,H from A,B overlaps with Y, D_2 and Y, D_1 of the granule C,D.

The complete optimization is handled in a distributed fashion using the Spark framework [19]. In this way, calculating the scores of each granule is performed very efficiently. A brief description about our optimization process has been given in Figure 4.8.

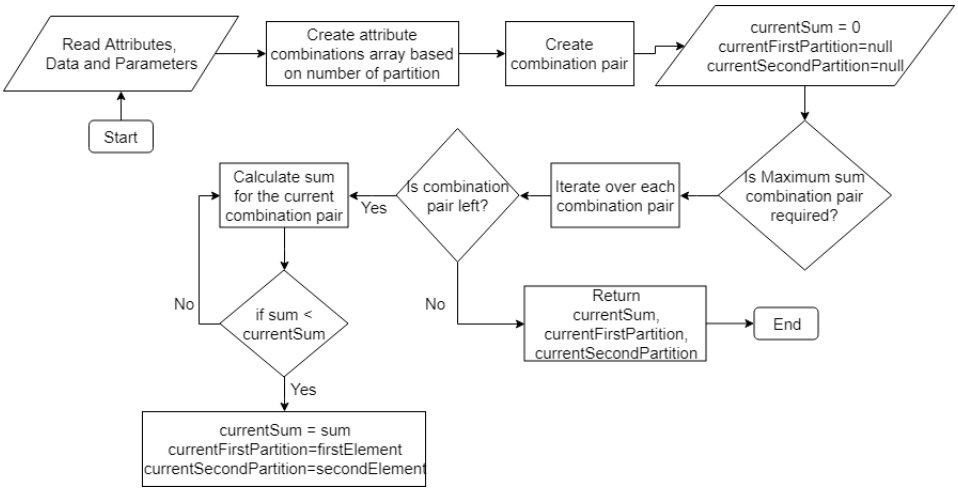


Figure 4.8: Data Distribution Strategy based on information granules

4.4.2.2 Privacy settings for Data Distribution

In addition to parameters that we assign for extracting action rules, we define parameters for privacy settings in our algorithms to manage privacy. The data that we

use for extracting actionable recommendations may comprise of sensitive information like user identifiers, places, and practices. Some of this information when placed together reveals greater privacy details to the user. In most of the cloud computing techniques, the data is distributed to multiple servers and the corresponding algorithms use such distributed data to mine or extract patterns. Since the data gets distributed to multiple servers, privacy of the data may get compromised and it is important to address such issues. Although, we do not handle such problems algorithmically, we give options to users to set input parameters, which protect the data privacy. These parameters are then used in our data partitioning module, to give certain attributes more importance, and protect them.

4.4.3 Semantic Data Distribution

In this section, we combine the methods proposed in Sections 4.4.1 and 4.4.2 in order to reduce the load of each worker in the cluster. To achieve this, we first partition the data by an attribute satisfying the needs of our proposed class attribute data partitioning method 4.4.1. Next, with each partitioned data, we split the attributes again satisfying the needs of our proposed vertical data distribution based on information granules method 4.4.2. Since one big data is broken into multiple chunks of small resilient distributed datasets, we can extract actionable patterns from all such small partitions in parallel. However, the small chunks may sometimes increase the complexity of our algorithms proposed in previous sections. For that reason, we propose the following load balancing approach to handle the situations, where the data has large number of attributes.

4.4.3.1 Load balancing parameter

Load balancing is another issue in cloud computing that we address in this work for extracting actionable recommendations. During very large data processing with distributed environment, it is required to give manageable work loads to the processing

unit to optimize throughput, scalability, and resource utilization [81]. Our method handles two levels of load balancing. Attracted from dynamic load balancing algorithms like Ant Colonization Optimization [82] and Honey Bee foraging [83], we propose load balancing that follows binary tree structure as given in Figure 4.9. We define a function that process this parameter and splits the data accordingly. This method is an extension of the vertical data partitioning methodology given in Section 4.4.2. Instead of terminating the data partitioning at depth 1, the data partitioning continues to depth **n** in a binary fashion, based on the assigned load balancing parameter. From the binary tree, each node is a given as a load to the action rule extraction algorithm and thus reducing the total load into tiny chunks. Assigning very high value (example: LoadBalancingParameter=13, with numberOfAttributes=12, each attribute would go into one partition) or very low value (example: LoadBalancingParameter=0, with numberOfAttributes=12), to this parameter increases complexity of extracting patterns. We recommend to set a mid-range value to this parameter. For all our experiments, we set this parameter as 2. The next level of load balancing is in-built load balancing functions in Apache Spark framework.

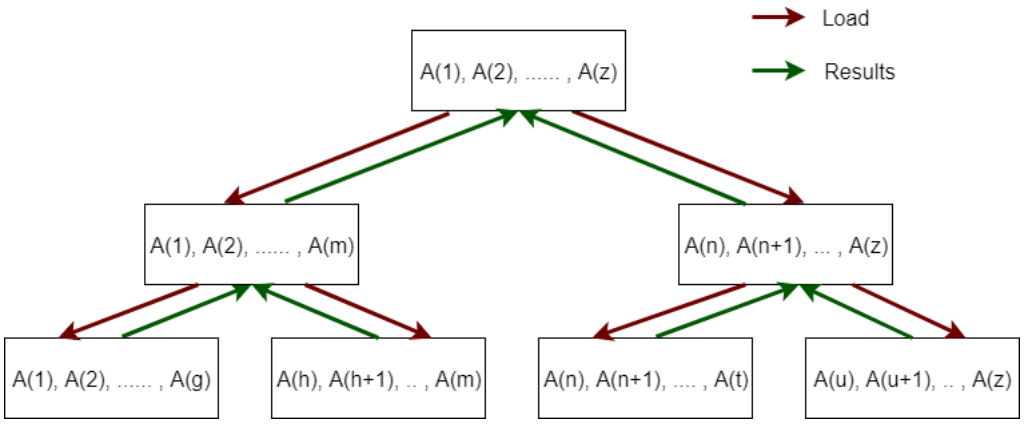


Figure 4.9: Binary Tree Load Balancing Strategy for Data Distribution

CHAPTER 5: DISTRIBUTED ACTION RULES OF LOWEST COST USING ACTION GRAPH

5.1 Extracting Actionable Patterns of Lowest Cost

Ras and Tzacheva [84] introduced the notion of cost and feasibility of Action Rules as an interestingness measure. They proposed a graph based method for extracting feasible and low cost Action Rules. Ras and Tzacheva [4] proposed a heuristic search of new low cost Action Rules, where objects supporting new set of rules also supports existing rule set but the cost of reclassifying them is much lower for new rules. Later, Tzacheva and Tsay [33] proposed tree based method for extracting low cost Action Rules.

Apart from Action Rules, some research has been done on extracting Actionable knowledge. For example, Yang, et.al [85] considered *Customer Attrition* in Customer Relationship Management (CRM) in telecommunications industry and the cost complexities involved in gaining profit to all customers. They proposed a method to extract low cost Actionable patterns for converting undesired customers to loyal ones while improve the net profit of all customers. Karim and Rahman [86] proposed another method to extract cost effective actionable patterns for customer attribtion problem in post processing steps of Decision Tree and Naive Bayes classifiers. Su, et.al [87] proposed a method to consider positive benefits that occurs by following an Action Rule apart from all costs that incur from the same rule. Cui, et.al [2] proposed to extract optimal actionable plans during post processes of Additive Tree Model (ATM) classifier. These actionable patterns can change the given input to a desired one with a minimum cost. Hu, et.al [88] proposed an integrated framework to gather cost minimal actions sets to provide support for social projects stakeholders

to control risks involved in risk analysis and project planning phases. More recently, Hu, et.al [89] developed a cost sensitive and ensemble framework to predict software project risk predictions and conducted large scale analysis over 60 models 327 real world project samples.

5.2 Action Graph

We build a graph called *Action Graph* from the Action Rules extracted using the SARGS algorithm as discussed in Section 5.3. We build Action Graph by using *atomic action terms* in Action Rules and their relation with other action terms. In general, graphs take the representation of $G = (V, E)$where V is a set of vertices and E is a set of edges connecting vertex pairs in V. All vertices and edges can contain their own properties that combined together uniquely represent vertices and edges respectively. We represent our Action Graph as an undirected graph $A_g = (A_v, A_e)$. In Action Graph, we treat action terms that we get from Action Rules as a set of vertices (A_v) and we create edge between a vertex pair $(a_m, a_n | a_m, a_n \in R_i)$, where R_i is an Action Rule. We set basic properties of an action term such as *Vertex Id*, *Name*, *Cost*, *Support*, *Neighbor Ids* and *Action Rules* of low cost based on the vertex as vertex properties of the Action Graph and *Co-occurrence Frequency* of a vertex pair as an edge property. Figure 5.1 gives a sample Action Graph for Action Rules extracted from Table 2.1 using the SARGS algorithm. For example red node means highest frequency, yellow node means medium frequency, and blue node means low frequency.

5.3 Action Graph Search Algorithms

Constructing an action graph from a set of action rules, we create three search algorithms for action graphs: *Dijkstra's Shortest Path*, *Breadth First Search*, and

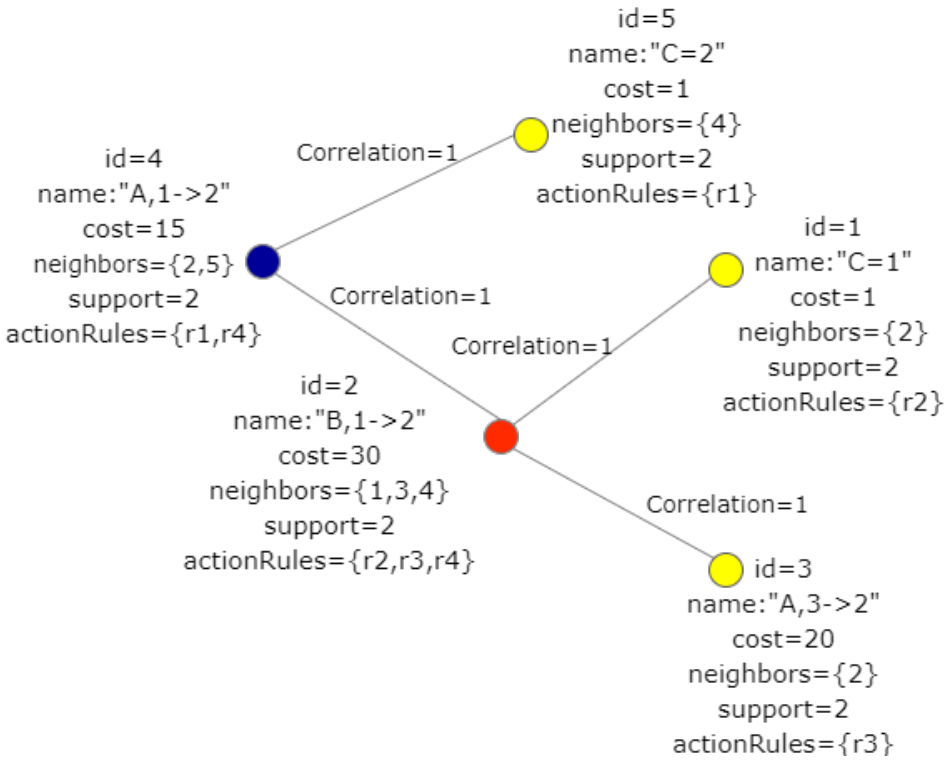


Figure 5.1: Sample Action Graph with Vertex Properties and Edge weights; Vertex color represents how frequently the action term occurs, with Red being the most frequent, and Yellow the least frequent.

Depth First Search to extract low cost actionable patterns [90].

5.3.1 Dijkstra's Shortest Path Algorithm

In the context of Action Graph, Dijkstra's shortest path algorithm works similarly to one given in Algorithm 1. However, introducing the notion of Cost of Action Rules, we change the Algorithm, as shown in Algorithm 12. That gives an overview on the Dijkstra's shortest path algorithm for Action Graphs. In Spark GraphX [48] library for processing large graphs, all nodes process their properties in parallel. Thus we consider following properties to each node in the graph:

- *vertexName, vertexCost*: corresponding vertex's name and cost respectively
- d: $(key, value)$ pair, where key represents the starting vertex(s) and $value$ consists of a path followed from s to the current node and corresponding path's

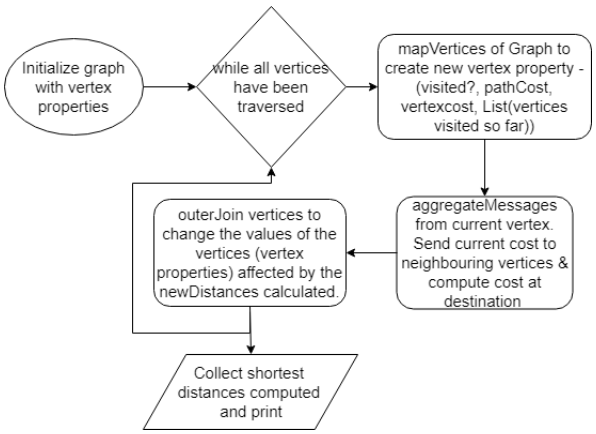


Figure 5.2: Dijkstra's algorithm flow chart for Action Graph

cost

Algorithm 12 gives an overview of our search algorithm with functions to send,receive and merge messages. The basic idea behind our search algorithm is very similar to *Dijkstra's shortest path algorithm* [91] adapted to distributed environment on cloud[90]. In each iteration: all vertices share their ActionTerm with its cost with their neighbors; all vertices add ActionTerms arriving from neighbors to their dictionary; all vertices combine the valid low cost ActionTerms with the ones already in their dictionary; the resulting ActionRules are sorted by cost in descending order; finally, all vertices share the set of low-cost ActionRules with their neighbors; algorithm runs for $n - iterations$, where n is the number of ActionTerms in the longest ActionRule, from the input list of ActionRules. The search algorithm takes the Action Graph $A_g = (A_v, A_e)$, where A_v is a set of vertices or ActionTerms and A_e is a set of edges connecting vertex pairs in A_v, and minimum cost threshold ρ. We send an initial empty message to start the functions. The first function to execute is the *ReceiveMsg()*. For better readability we explain in the order of *SendMsg()*,*MergeMsg* and *ReceiveMsg()*. *Steps 6-10* gives procedure to do for all vertices when they need to send a message to their immediate neighbors. Each vertex process each edge originating from them. For each available low cost Action Rule r, it checks if

Algorithm 12 Dijkstra's Shortest Path algorithm for Action Graphs

Require: $A_g = (A_v, A_e)$, a source vertex u and cost threshold ρ

A'_g := A_g.mapVertices(v => (v.vertexName,v.vertexCost,d)

procedure SENDMSG(id,*srcVertex*,*dstVertex*)

    sources := Collect sources from srcVertex.d that are not available in dstVertex.d

    return a dictionary with *sources* to *dstVertex*

procedure MERGEMSG(m1,m2)

    mergedMessage := $\emptyset$

    for *source* $\in$ $m1$.*sources* **do**

        if m1.source.cost < m2.source.cost **then**

            mergedMessage.source := m1.source

        else

            mergedMessage.source := m2.source

    return mergedMessage

procedure RECEIVEMSG(id,oldProp,newProp)

    for *source* $\in$ *newProp* **do**

        newCost := Add *this.name*,*this.cost* to newProp.source

        oldProp.source = newCost

    return oldProp

A_g^{final} := A'_g.aggregateMessages(*SendMsg*, *MergeMsg*, *ReceiveMsg*)

return all paths and costs from all vertexes

$r \subseteq dstn.neighbors$ in Step *9*. This step filters the dictionary in each vertex remove ActionTerms that are irrelevant to the destination vertex . To avoid duplicate rules from multiple vertices, we send only the combination of ActionTerms that are new to the destination vertex. In *Steps 11-13* we give a procedure for each vertex to combine messages from multiple vertices. This function simply combines all messages (dictionaries of ActionTerms with their Costs) and into a single Dictionary. This single Dictionary is processed via the *ReceiveMsg()* function for processing. In *Steps 1-5* we show the processing the *ReceiveMsg()* performs - for all vertices when they receive a message. When a vertex receives a set of ActionTerm combinations and their corresponding costs, it adds its own cost to produce a Low Cost Action Rule. If the total cost is less than or equal to the given cost threshold ρ, the vertex adds the Action Rule to its list of Low Cost Action Rules. The main function is described in *Step 16*, where we initiate the first *messageSend()* operation to $v \in A_v$. First, we populate *ActionRules* property of each vertex to the combination of current vertex and its immediate neighbor and respective cost. Next, all vertices send an empty message to all their immediate neighbors. This continues for n iterations as mentioned above. Once all iteration are over, we obtain an Action Graph A'_g containing Action Rules along with their cost for each vertex. We then sort the rules by cost in Descending Order, and suggest to the user the top 5 lowest cost rules for each vertex. The top 5 lowest cost Action Rules from all vertices form the set of the discovred Action Rules of Lowest Cost.

In summary: for Algorithm 1, in $SENDMSG$ function, we choose the sources that the destination is not having and send paths and costs of only sources that are not available in the destination. In $MERGEMSG$ function, for each source we select a path with minimum cost and in $RECEIVEMSG$, we receive all messages and add current node's cost and update graph properties. By following these functions,

eventually paths and costs propagates to all nodes in the graph. By the end of $n/2$ iterations, all nodes would have least cost to reach from source to themselves.

5.3.2 Breadth First Search Algorithm for Action Graph

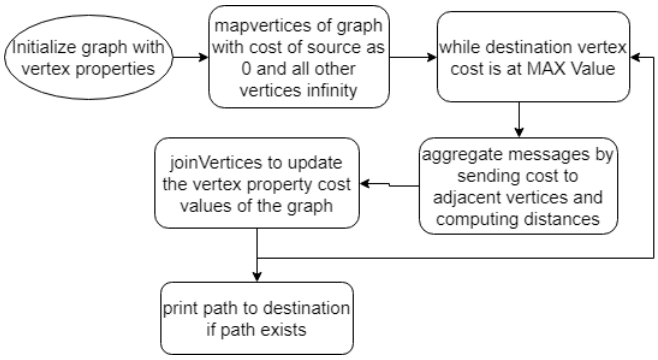


Figure 5.3: Breadth First Search algorithm flow chart for Action Graph

Since maintaining a queue to track the traversal is complex in parallel computing engines like Spark GraphX, we propose modified strategies for *Breadth-First* and *Depth-First* searches in Action Graphs. BFS works alike Algorithm 12 with one exception. Instead of choosing a path with minimum cost in the $MERGEMSG$ function, for each source vertex we choose a path from the latest source. For example, if vertexes 1 and 2 are sending path and cost for the source node 3 to vertex 4, the $MERGEMSG$ function of vertex 4 chooses path and cost of source node 3 from vertex 2. Once this entry is updated, it cannot be altered in future but it can propagate to update its neighbor entries [92].

5.3.3 Depth First Search algorithm for Action Graph

Depth First Search (DFS) is one of the more computationally complex problems ($P - Complete$) to be incorporated into parallel frameworks. For long time in the literature, parallelizing DFS is one of the main concerns and several variations of DFS has been proposed [93] [94]. In this work, we propose DFS for Action Graphs to extract low cost Action Rules as given in Algorithm 13. For the sake of Spark GraphX

framework, we attach following parameters to each node for the algorithm [92]:

- *vertexName, vertexCost*: corresponding vertex's name and cost respectively
- *neighborPath*: path followed by a node to traverse among immediate neighbors
- l: Similar to dictionary d in Algorithm 12. We also attach which node to visit next along with path and cost

Thus each vertex share their *neighborPath* in the first iteration with their neighbors(but only to specified neighbor vertex get the content). In the remaining iterations $SENDMSG$ function sends the dictionary l. Unlike *Dijkstra's* and *BFS*, we are not gathering paths for all possible source and destination pairs. Instead, we are setting each vertex as a source vertex and collecting a path using DFS traversal to reach all other vertexes. Thus in the $MERGEMSG$ function, we are getting updated pat and cost from neighbors. In $RECEIVEMSG$ function, we simply find nodes, from *neighborPath*, that are not visited and attach them to the path in same sequence and update the cost and next node to visit parameters.

5.3.4 Post-processing of Low Cost Action Rules

By following the Algorithm 12 or 13, we obtain all low cost Action Rules. Some Action terms in Action Rules may have high correlations. We propose a method to reduce further the cost of the obtained rules by considering edge weights in our *Action Graph* during post-processing steps. We assign edge weights between two vertices or action terms based on their frequencies of co-occurring together in Action Rules. We define a correlation threshold θ to check if two action terms in an Action Rule is highly correlated. We assume that two action terms $a_{r1}, b_{r1}|(a_{r1}, b_{r1}) \in r1$, where $r1$ is an Action Rule, to be highly correlated if their co-occurring frequency w is greater than or equal to θ. We propose that when two action terms satisfy the $w \geq \theta$, then the action suggested by the first term is expected to trigger the action suggested by

Algorithm 13 Depth First Search algorithm for Action Graphs

Require: $A_g = (A_v, A_e)$, a source vertex u and cost threshold ρ

A'_g := A_g.mapVertices(v => (v.vertexName, v.vertexCost, neighborPath, l)

procedure SENDMSG(id,*srcVertex*,*dstVertex*)

    return *dstVertex.l*

procedure MERGEMSG(m1,m2)

    mergedMessage := ∅

    for *source* ∈ *m*1.*sources* **do**

        if |m1.source| > |m2.source| **then**

            mergedMessage.source := m1.source

        else

            mergedMessage.source := m2.source

    return mergedMessage

procedure RECEIVEMSG(id,oldProp,newProp)

    for *source* ∈ *newProp.msg* **do**

        if newProp.source.target = id **then**

            newPath = newProp.source.path

            for *node* ∈ *oldProp.neighborPath* **do**

                if *node* ∉ *newProp.source.path* **then**

                    newPath := Add (node.name,node.cost)

            if *allVertices* ∈ *newPath.names* **then**

                target := ∅

            else

                target := newPath.last.name

            oldProp.source := (newPath,target)

    return oldProp

A_g^{final} := A'_g.aggregateMessages(*SendMsg*, *MergeMsg*, *ReceiveMsg*)

return all paths and costs from all vertexes

	(a,3->2)	(b,1->2)	(c=1)
(a,3->2)	0		
(b,1->2)	1	0	
(c=1)	0	1	0

Figure 5.4: Example Correlation Matrix of the action term $(b, 1- > 2)$

the second one. Therefore, the lowest cost action can be dropped from the total cost. For each vertex, we define a correlation matrix, which gives correlation frequency between the current vertex or action term and its neighbor. Figure 5.4 gives a sample correlation matrix for the action term vertex $(b, 1- > 2)$. With this correlation matrix, we can identify which 2 terms are highly correlated. Then we process each Action Rule from the dictionary of low cost Action Rules of the current vertex. When a highly correlated pair occurs in the Action Rule, we drop the cost of lowest cost action term. For example, cost of the Action Rule $(b, 1- > 2) \cap (c = 1)$ can be reduced from *31* to *30*, if the correlation threshold θ is set to *1*.

All the above methods use random distribution of data. Either Hadoop or Spark divides the data based on its block size, and assigns a random part of the data to each data processing node. In the next section , we propose a more intelligent method for distribution of the data, in order to accomplish more accurate support, confidence, utility, and coverage measures for the computed Action Rules.

CHAPTER 6: EXPERIMENTS AND RESULTS

To experiment on algorithms discussed in the previous chapter, we make use of datasets available in the machine learning repository of the Department of Information and Computer Science of the University of California, Irvine [95]. Out of many datasets, we used Car Evaluation data and the Mammographic Mass data. These datasets are more suitable for rule-based classification algorithms and they have categorized attributes. Since these datasets are relatively small in size, in order to test them for scalability with the proposed distributed processing algorithms, we replicate their data rows 1024 and 2056 times respectively for CarEvaluation and MammographicMass datasets, in order to increase data size. We test our algorithms on both replicated and non-replicated data for CarEvaluation and Mammographic datasets.

6.1 Description of Datasets

6.1.1 Car Evaluation Data

The Car Evaluation Data consists of records describing a car's goodness and acceptability.

The Car Evaluation dataset [95] is donated by Prof. Dr. Marko Bohanec, from Department of Knowledge Technolgoies, Jozef Stefan Institute, in Liublijana, Slovenia. It is intended to evaluate cars according to the car acceptability, according to its buying price, maintenance cost, technical characteristics such as comfort, number of doors, number of persons to carry, the size of its luggage boot, and the car safety. The Car Evaluation dataset has 1728 tuples, and 7 attributes.

For large scale testing, we replicated this dataset first 512 times, and then 1024 times.

6.1.2 Mammographic Mass Data

Mammographic is the most effective method for screening breast cancer.
The Mammographic-Mass dataset [95] is donated by Prof. Dr. Rdiger Schulz-Wendtland from the Institute of Radiology at the University Erlangen-Nuremberg, Germany. This dataset is used to predict the severity (benign or malignant) of a mammographic mass lesion from BI-RADS attributes and the patient's age. It contains a BI-RADS assessment, the patient's age and three BIRADS attributes together with the ground truth (the severity field) for 516 benign and 445 malignant masses that have been identified on full field digital mammograms collected at the University Erlangen-Nuremberg. The Mammographic-Mass dataset contains 961 instances, and has 6 attributes.
For large scale testing, we replicated this dataset first 1024 times, and then 2048 times.

6.1.3 Charlotte Business Data

We also test with the city of Charlotte North Carolina BusinesWise data , which is donated by the Charlotte Chamber of Commerce. This data collects details of over 20,000 business companies in Mecklenburg county, North Carolina. The data includes their City, StartYear, Sector, Specialization of the company in a selected sector, SiteType, Employees count at the site, Total employees in the company including all branches, Site building type, Total sites and Estimated Sales. From this data, our focus is how to increase a company's estimated sales from < 2 million US dollars to the range between 3 million and 10 million US dollars.

6.1.4 Net Promoter Score Data

The NPS (Net Promoter Score) [96] dataset is collected customer feedback data related to heavy equipment repair. The entire dataset consists of 38 companies, located in different sites across the whole United States as well as several parts of

Canada. Overall, there are about 340,000 customers surveyed in the database over time span of 2011-2015. Customers were randomly selected to answer a questionnaire which was specifically designed to collect information relevant to NPS (structured into so-called "benchmarks"). All the responses from customers were saved into database with each question (benchmark) as one feature in the dataset. Benchmarks include numerical scores (0-10) on different aspects of service: e.g. if job done correctly, are you satisfied with the job, likelihood to refer, etc. The dataset also contains customer details (name, contact, etc.) and service details (company, invoice, type of equipment repaired, etc.). The decision attribute in the dataset is *PromoterStatus* which labels each customer as either *promoter*, *passive* or *detractor*. The decision problem here is to improve customer satisfaction / loyalty as measured by Net Promoter Score. The goal of applying Action Rules to solve the problem is to find minimal sets of actions so that to "reclassify" customer from "Detractor" to "Promoter" and the same improve NPS. Due to confidential details in the data, we used some of this data only in few of our evaluations. The given dataset contains three decision attribute values: *Promoter*, *Passive* and *Detractor*. Figure 6.1 gives decision value **Promoter Status** distribution for the four given datasets. Each dataset represents a business at a particular location.

6.1.5 HCUP - Hospital Readmission Data

In this work, we experiment with a Medical domain dataset called: Healthcare Cost and Utilization Project(HCUP) data. HCUP offers multiple data for analysis such as *Inpatient databases*, *Emergency department databases*, and *Readmission databsaes* at both state level and national level. However, the national level data is a weighted sample of data collected from all state level datasets. For our analysis we use the *State Inpatient Data* (SID) of the state Florida of years 2010-2012 [97]. All our data are organized as each data record representing a patient's hospital visit and each patient visit has 298 attributes. Table 6.1 gives a brief description about interesting

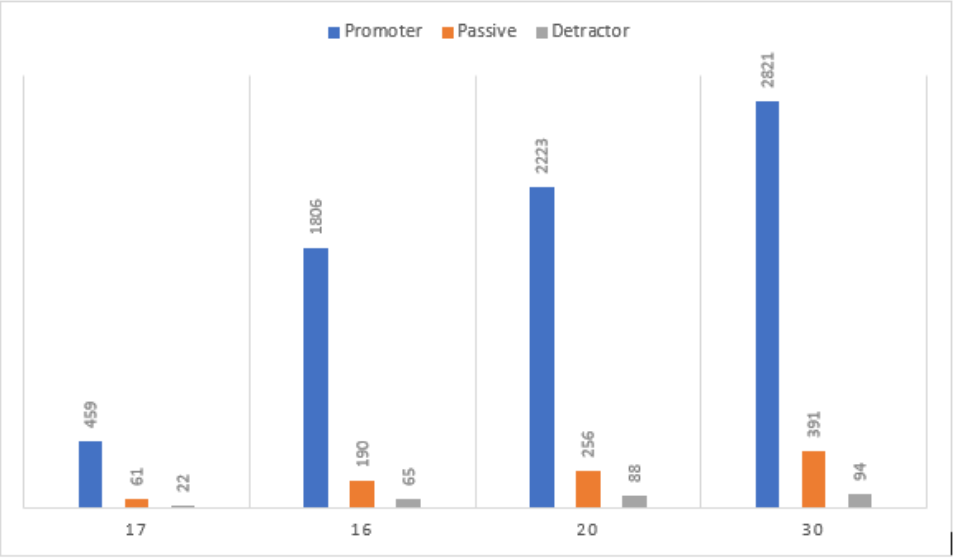


Figure 6.1: Number of instances in each Customer Type

Table 6.1: Interesting attributes in the datasets

Attribute Name	**Attribute Description**
Age	Patient's age during discharge
Died	Boolean indicator of whether a patient died during the hospital stay
DX, DXCCS	Diagnosis codes representing all diagnosis that a patient follows during their hospital visit
PR, PRCCS	Procedure codes representing all procedures that are followed on a patient during their hospital stay
LOS	Length of hospital stay
VisitLink	Identifier of a patient
Race	Race of a patient
DaysToEvent	Number of days before next admission

attributes that are available in our datasets and that we use in all our methods. In total, our SID dataset has 4,008,182 records, representing patient visits, out of which 2,625,083 are unique patients. In this work, we augment the HCUP dataset, by adding a new Feature - called *Readmitted* attribute. We create this attribute by calculating its value for each patient visit using attributes *LOS* and *DaysToEvent*. Based on this measured *Readmitted*, attribute, we give a plot on number of patients

with their corresponding number of hospital readmission in Figure 6.2. In this figure, we can see that the number of patients who have been readmitted to the hospitals at very low frequency decreases constantly and have sudden spikes between readmission frequencies 40 and 60, and becomes constant for higher frequencies. We also note that almost 72% patients in the data has atleast been readmitted to the hospital at least once.

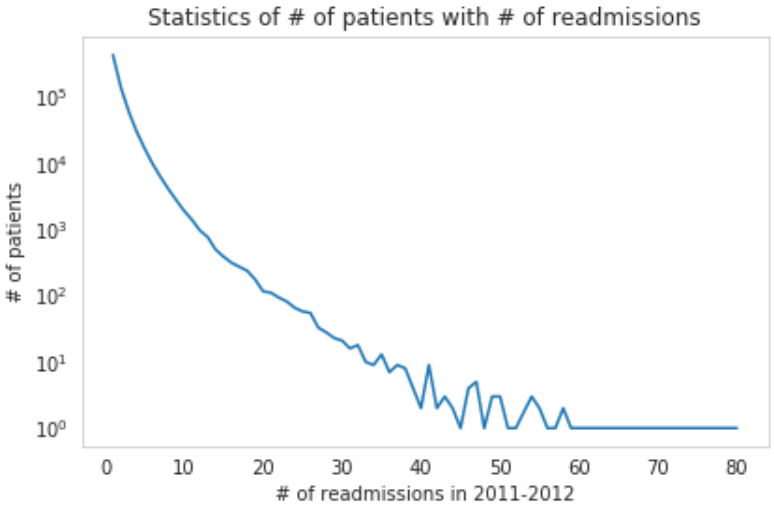


Figure 6.2: Representation of number of patients in Florida with their corresponding readmission frequency in 2011-2012

Out of 298 attributes, 70% of the attributes are allocated to mark diagnoses and procedures that a patient follows during their hospital stay. Most importantly, these diagnoses and procedures are represented as *ICD-9-CM* (International Classification of Diseases, Ninth Revision, Clinical Modification) codes. The data reports two varieties of diagnoses and procedures for covering these ICD-9-CM codes. One is simple ICD-9-CM code, which has around 8,900 unique codes in all diagnoses and procedures. And, the other is an aggregated version of these codes, defined by *Clinical Classification Software* (CCS). These codes classifies diseases to their major classification and thus we have only around 520 unique ICD-9-CM codes from these attributes. In all our experiments, we use the later versions of codes to improve efficiency of algorithms

and robustness in recommendations.

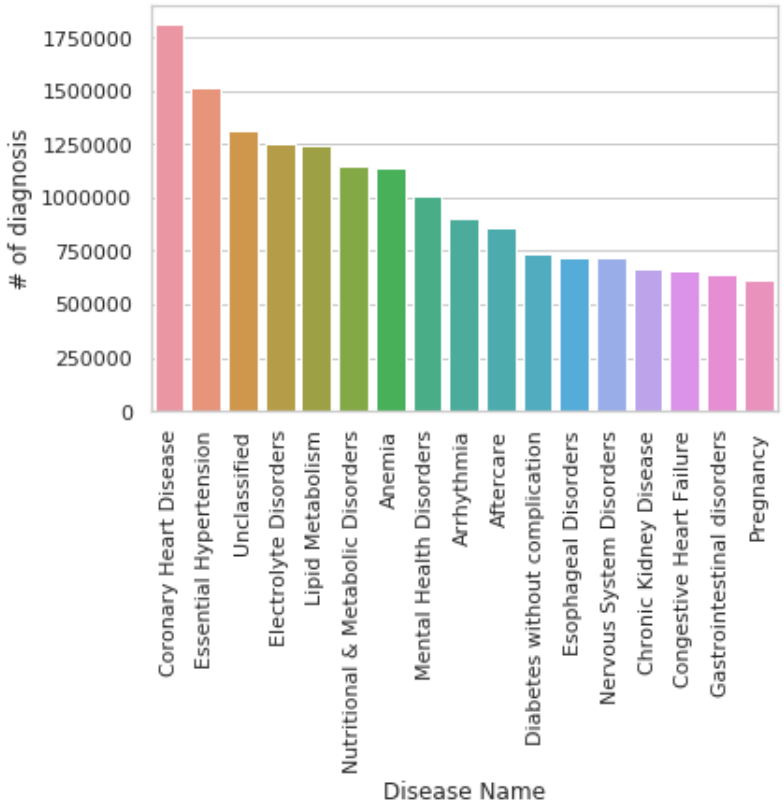


Figure 6.3: Top 15 diagnosed diseases for patients in Florida based on their frequency in 2011-2012

In Figures 6.3 and 6.4, we represent diseases that have been most diagnosed and procedured in Florida in years 2011 and 2012. It is interesting to note that the most diagnosed disease(*Choronary Heart Disease*) is not present in the top procedures list and similarly, the top procedure(*Benign Neoplasm*) is not present in top diagnosis.

According to our data, almost 12% of patient visits resulted in death after following certain procedures to cure diseases. In Figure 6.5, we show the top 15 diseases that result in patient's death during their procedure to cure the disease. From this figure, we can note that these 75% of diseases correlates with top diseases for which the patients have undergone procedures as given in the Figure 6.4.

We also evidence that our datasets are rich in diversity. Our dataset comprise of details about 58% female patient and 42% female patients. We show the diversity in

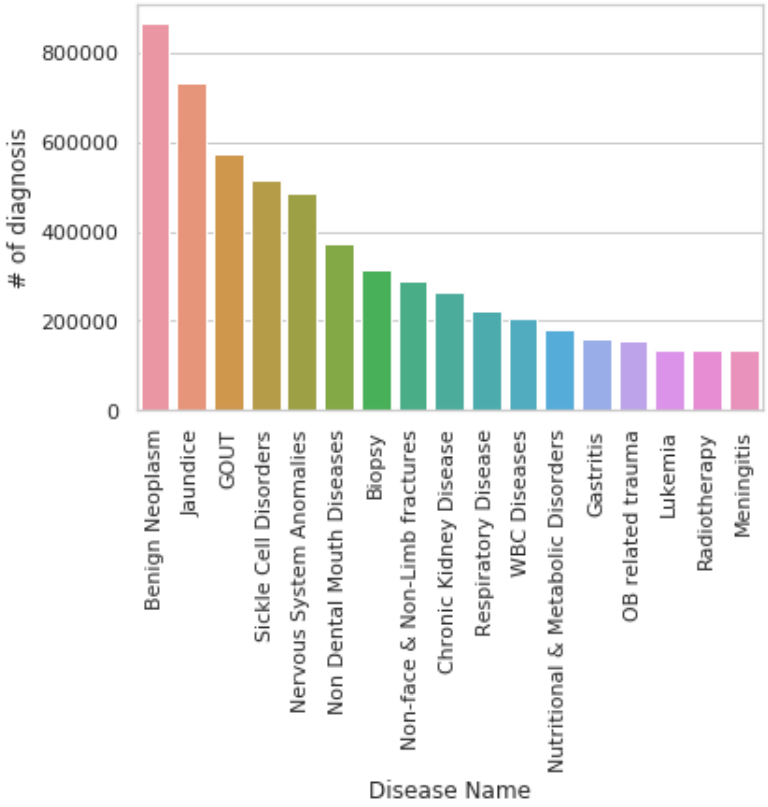


Figure 6.4: Top 15 procedured diseases for patients in Florida based on their frequency in 2011-2012

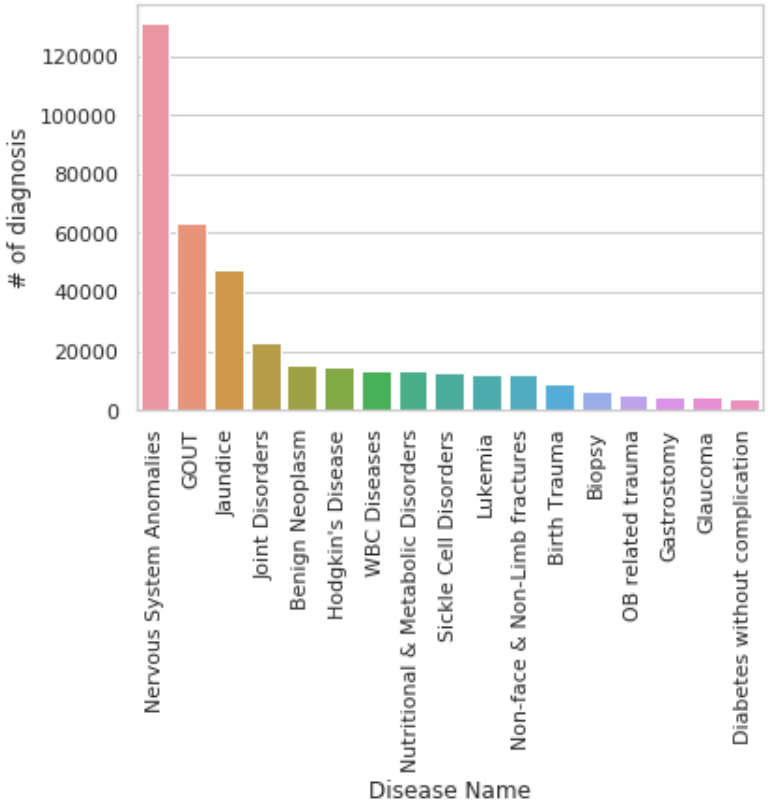


Figure 6.5: Top 15 diseases that caused death for patients in Florida during their hospital admission in 2011-2012

patients race in the Figure 6.6. From this figure, we note that our dataset has a good representaion of white, black, and hispanic patients.

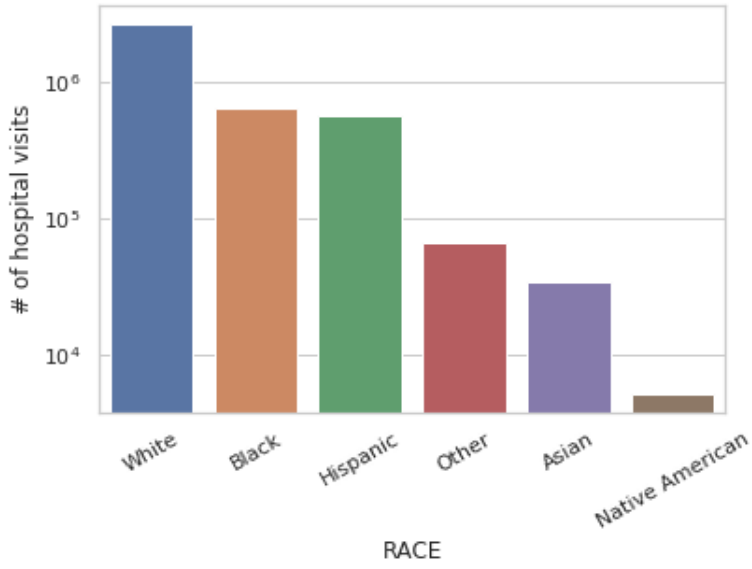


Figure 6.6: Distribution of diversity of patients and their corresponding admission rates in Florida hospitals during 2011-2012

6.1.5.1 Hospital Readmission Reduction

There are few works that use data mining techniques to study the increasing hospital readmissions problem. Unplanned hospital readmissions are expensive for both patients and healthcare, and they create unfortunate outcomes to everyone (patients, physicians, tax payers, and healthcare systems) [98]. After the advent of Hospital Readmission Reduction Program in Affordable Care Act, the hospital readmission in the country declines moderately, but still creating new challenges to hospitals [99]. Recently various domains like medicine [100], education [101], and business [102] started adopting data science research in their respective problems. Many research studies have focused on using the voluminous real world datasets for healthcare applications and decision making using such data mining and knowledge extraction techniques [103]. For example, in particular to hospital readmission, researchers create a machine learning model to predict patient readmissions using just billing codes and

basic patient admission characteristics [104]. Some focus on predicting the liklihood of patient readmitting to the hospital, modelled as risk prediction, using Support Vector Machines, Random Forests, and Neural Networks [105]. Similarly, there is a study on using logistic regression to measure the relationship between diabetes and early readmission [106], and a study on using a classic data mining technique like Support Vector Machine to predict readmission [107] using other features such as patient demographics, disease type, admission type, and clinical procedures undertaken. Recently, there is an interesting study on designing a personalized procedure graphs, which gives a probability on patient's future procedure and recommend hospitals in making decisions for a patient [21, 108].

6.2 Experiment System Configuration

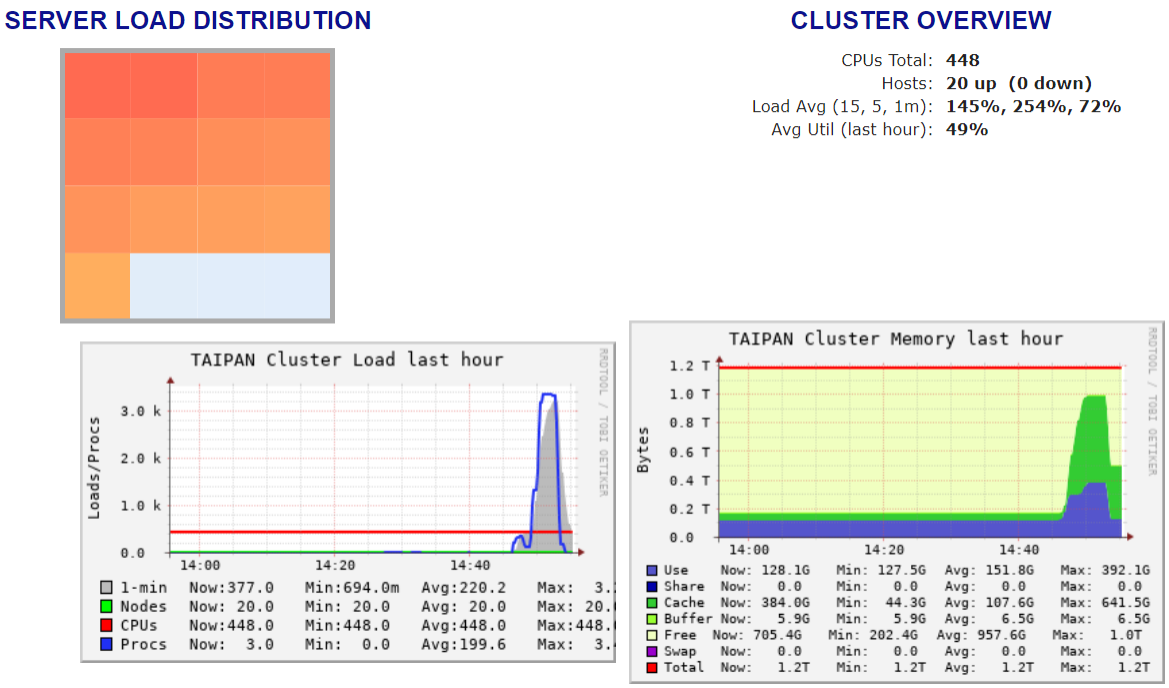


Figure 6.7: Hadoop cluster statitics - load distribution, active node status, cluster load, and cluster memory

We conduct all our experiments in the University of North Carolina at Charlotte's

Hadoop research cluster (URC). The cluster comprise of 4 master nodes and 16 slave nodes, each node with 87 tera-bytes of Hadoop Distributed File System(HDFS) available to the research faculty and students. Each computer core consists of 12 cores, totaling upto 192 computer cores in the cluster. Each node is dual Intel 2.93GHz 6-core processor with 64GB RAM. The Hadoop cluster uses an open source job monitoring system named *Ganglia* to monitor the hardware performance and jobs that are running in the cluster. We show these monitoring systems in Figures 6.7 and 6.8. Figure 6.7 illustrates the monitoring system that presents the load distribution within nodes in the cluster, number of active/inactive nodes, average load and average resource utilization, total loads/processes, and cluster memory usage. Figure 6.8, illustrates the monitoring system that presents the CPU utilization of the job in the cluster, total networking activity in the cluster for the job(data transfers between nodes), and load for each node in the cluster.

6.3 MR Random Forest Experiment - in Hadoop MapReduce

We used two datasets for testing our proposed MR - Random-Forest algorithm for distributed action rules discovery: Car Evaluation dataset and Mammographic-mass dataset, obtained from the Machine Learning Repository by Information and Computer Sciences of the University of California, Irvine [95].

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

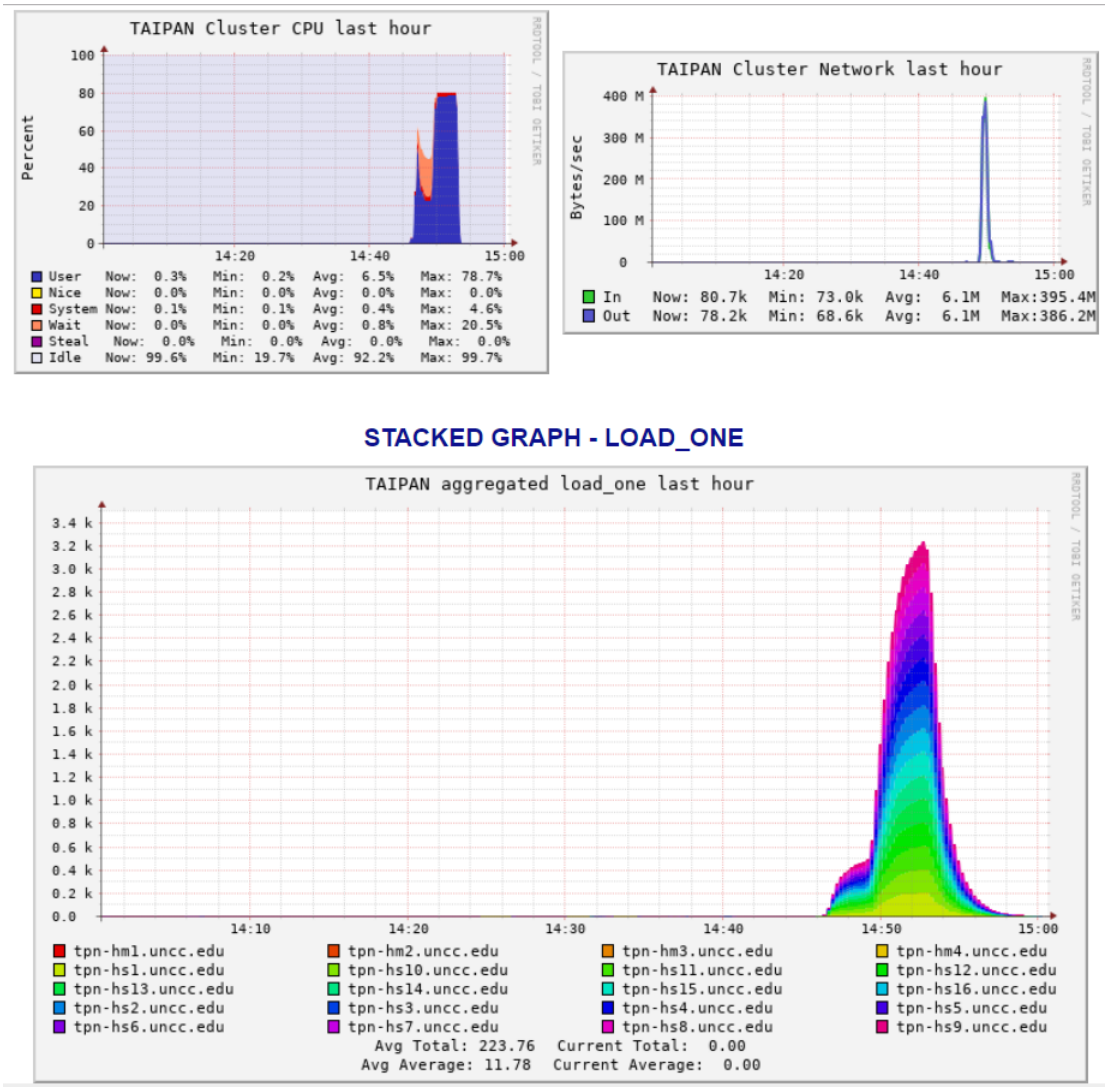


Figure 6.8: Hadoop cluster statitics - CPU usage, network activity, and load/node

posed algorithm in a distributed environment. This also brings the final dataset to size greater than 1.04 MB, so Hadoop splits it automatically.

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

Table 6.2 shows complete properties, like number of instances, replication factor and data size, of the above discussed datasets.

Table 6.2: Properties set for Car Evaluation dataset and Mammographic-Mass Dataset for MR-Random Forest and MR-Apriori Algorithms

Property	**Car Eval. Data**	**Mammo. Mass Data**
Number of instances	1728	961
Rplication Factor	116	518
Number of instances after replication	200448	497798
Attributes	7 attributes -Buying -Maintenance -Doors -Persons -Luggage Boot -Safety -Class	6 attributes -BI-RADS -Patient's age -Shape -Margin -Density -Severity
Decision attribute values	Class (unacc, acc, good, vgood)	Severity (0 - benign, 1- malignant)
Original data size	52KB	16KB
Data size after replication	5.922MB	7.93MB

The Car Evaluation dataset [95] is donated by Prof. Dr. Marko Bohanec, from Department of Knowledge Technolgoies, Jozef Stefan Institute, in Liublijana, Slovenia. It is intended to evaluate cars according to the car acceptability, according to its buying price, maintenance cost, technical characteristics such as comfort, number

of doors, number of persons to carry, the size of its luggage boot, and the car safety. The Car Evaluation dataset has 1728 tuples, and 7 attributes, as shown in Table 6.2. For the purpose of this study, the Car Evaluation dataset was replicated 116 times, in order to increase its size, and demonstrate the scalability of our proposed method. Action Rules extracted for this dataset can suggest actions to be undertaken (changes in flexible attributes) if the user would like to increase the car's safety, or if the user would like to change the car state from 'unacceptable' (unacc) to 'acceptable' (acc). An example Action Rule extracted from this dataset is:

$$ar_{Car1}\big(class, unacc \rightarrow acc\big) = \big(buying, buyinglow \rightarrow buyinglow\big) \wedge$$
$$\big(persons, persons2 \rightarrow persons4\big) \wedge \big(safety, \rightarrow safetyhigh\big) \longrightarrow \big(class, unacc \rightarrow acc\big)$$
$$[Support : 237 \& Confidence : 93.0\%]$$

The rule ar_{Car1} means that: if the buying price of the car remains low (*buyinglow*), and the number of persons it can carry increases from 2 (*persons2*) to 4 (*persons4*), and the safety of the car increases from any value to high (*safetyhigh*), then the decision attribute (*class*) value is expected to change from unacceptable (*unacc*) to acceptable (*acc*). A total of 237 tuples (*objects*) support this rule, and we are 93% confident in the validity of this rule. Example Actions, called Meta-Actions, which can trigger the above changes are: 'improve air bags' (to increase safety); 'improve breaks' (to increase safety); 'make larger salon' (to increase person capacity of the vehicles). These are called Meta-Actions as described by Tzacheva and Ras [35], since they trigger the suggested changes in flexible attributes specified by the Action Rules. The Meta-Actions can either be provided by expert in the domain and added to the original data to augment it, or they can be automatically extracted from text descriptions associated with the data as shown by Kuang and Ras [20]. For this study, the attributes *Buying, Maintenance, Doors* are designated as *Stable Attributes*, and the attributes *Persons, LuggageBoot, Saftety* are designated as Flexible Attributes, and the attribute Class is designated as the decision attribute, which is also a flexible

attribute. These parameters are shown in Table 6.

The Mammographic-Mass dataset [95] is donated by Prof. Dr. Rdiger Schulz-Wendtland from the Institute of Radiology at the University Erlangen-Nuremberg, Germany. This dataset is used to predict the severity (benign or malignant) of a mammographic mass lesion from BI-RADS attributes and the patient's age. It contains a BI-RADS assessment, the patient's age and three BIRADS attributes together with the ground truth (the severity field) for 516 benign and 445 malignant masses that have been identified on full field digital mammograms collected at the University Erlangen-Nuremberg. The Mammographic-Mass dataset contains 961 instances, and has 6 attributes, as shown in Table 5. For the purpose of this study, the Car Evaluation dataset was replicated 518 times, in order to increase its size, and demonstrate the scalability of our proposed method. Action rules extracted from the Mammographic-Mass dataset can suggest actions to be undertaken (changes in flexible attributes), in order to re-classify a mammographic mass lesion (tumor) from class: malignant to class: benign. An example Action Rule extracted from this dataset is:

$ar_{Mam1}(severity, 1 \rightarrow 0) = (Margin, 3 \rightarrow 4) \wedge (BI-RADS, 5 \rightarrow 4) \wedge (Density, \rightarrow 3) \Longrightarrow (severity, 1 \rightarrow 0)[Support: 284 \& Confidence: 82.4\%]$

The rule arMam1 means that: if the Margin of the lesion (*tumor*) changes from 3 to 4, and the *BIRADS* assessment changes from 5 to 4, and the Density of the lesion (*tumor*) changes from any value to 3, then the *severity* (decision attribute) is expected to change from value 1 (malignant) to value 0 (benign). A total of 284 tuples (objects) support this rule, and we are 82.4% confident in the validity of this rule. The suggested desired changes can be triggered by Meta-Actions [35]. Example Meta-Actions, which can trigger the above changes are: 'doctor prescribes specific medication' (to change BI-RADS assessment); or 'doctor performs a specific medial procedure' (to change the margin of the lesion). For this study, we designate *BIRADS, Margin, Density, Shape* as Flexible Attributes. We designate *Shape, Age* as a Stable

Attributes. We designate Severity as our decision (class) attribute, which is also a flexible attribute. These parameters are shown in Table 6.3.

Table 6.3: Parameters used for Action Rules discovery on the Car Evaluation dataset and Mammographic- Mass dataset for MR-Random Forest and MR-Apriori Algorithms

Property	**Car Eval. Data**	**Mammo. Mass Data**
Stable attributes	Maintenance, Buying Price, Doors	Age, Shape
Required decision action	(Class) $unacc \rightarrow acc$	(Severity) $1 \rightarrow 0$
Minimum Support α and Confidence β	150, 80%	50, 70%

Since we replicated the datasets multiple times, as shown in Table 6.2, the size of the data was substantially increased from the original. Next, we ran our experiment, and Hadoop made 6 splits of the data for the Car Evaluation dataset, and it made 8 splits of the data for the Mammographicmass dataset. The ARoGS algorithm took 1.84 minutes to process the Car Evaluation data on a single node, and it took 1.12 minutes to process the Car Evaluation dataset on 6 nodes. The Association Action Rules algorithm took 11.09 minutes to process the Car Evaluation dataset on a single node, and it took 5.4 minutes to process the Car Evaluation dataset on 6 nodes. The ARoGS algorithm took 0.53 minutes to process the Mammographic Mass dataset on a single node, and it took 0.29 minutes to process the Mammographic Mass dataset on 8 nodes. The AAR algorithm took 9.4 minutes to process the Mammographic Mass dataset on a single node, and it took 5.4 minutes to process the Mammographic Mass dataset on 8 nodes. A comparison of the processing time for these algorithms is shown on Table 6.4.

The processing times shown in Table 6.4. indicate that: the larger the data size is, the faster our algorithms run (both ARAS and AAR algorithms), when using multiple nodes (in a distributed environment with MapReduce framework), compared to a

Table 6.4: Comparison of processing time for ARAS and AAR algorithms using MapReduce method on Hadoop

Dataset	**# of splits (nodes)**	**ARAS(mins)**	**AAR(mins)**
Car Evaluation Data	1	1.84	11.09
	6	1.12	5.4
Mammographic Mass Data	1	0.53	9.4
	8	0.29	6.2

single node (a single machine). From the results in Table 6.4, we can also see that ARAS algorithm generates the Action Rules much faster than the AAR algorithm does, while using the MR - Random Forest method in the Reduce phase for both. The AAR (Association Action Rules) takes a much longer time to generate Action Rules because it follows Apriori-like method described in section 3.3 to produce all possible combination of action sets and from these action sets, it generates all possible Action Rules. Table 8. depicts sample comparison of rules generated by both the algorithms on the Car dataset.

Next, we compare the ARAS and the AAR algorithm. Our results indicate that the ARAS algorithm produces more general Action Rules, while the AAR algorithm produces more specific Action Rules. By general Action Rule we mean that the rule contains an atomic action set like $(safety, \rightarrow safetyhigh)$ i.e. the *safety* is changed from any value to value *safetyhigh*. On the other hand, the AAR algorithm produces only specific Action Rules i.e. the action sets have both values chageFrom and changeTo specified, such as: $(safety, safetlylow \rightarrow safetyhigh)$. Even though the AAR algorithm follows Apriori-like method and takes much longer time to process, it generates more rules comparing to the ARAS method. For our study, the ARAS produced 20 Action Rules the Car Evaluation Dataset, while AAR produced 124 Action Rules, out of which 80 rules can be generalized to the rules produced by ARAS algorithm. We show an example of ARAS general Action Rule, and its corresponding

AAR specific Action Rules on Table 6.5.

Table 6.5: Comparison of general and specific Action Rules produced by ARAS and AAR respectively

ARAS	**AAR**
$(safety, \rightarrow safetyhigh)$ $(buying, buyinglow \rightarrow buyinglow) \wedge (maint, maintvhigh \rightarrow maintvhigh) \wedge (persons, persons2 \rightarrow persons4) \wedge (safety, \rightarrow safetyhigh) \Longrightarrow (Class, unacc \rightarrow acc)[Support : 232\&Confidence : 100.0\%]$	$(buying, buyinglow \rightarrow buyinglow) \wedge (maint, maintvhigh \rightarrow maintvhigh) \wedge (persons, persons2 \rightarrow persons4) \wedge (safety, safetylow \rightarrow safetyhigh) \Longrightarrow (Class, unacc \rightarrow acc)[Support : 232\&Confidence : 100\%]]$
	$(buying, buyinglow \rightarrow buyinglow) \wedge (maint, maintvhigh \rightarrow maintvhigh) \wedge (persons, persons2 \rightarrow persons4) \wedge (safety, safetymed \rightarrow safetyhigh) \Longrightarrow (Class, unacc \rightarrow acc)[Support : 232\&Confidence : 100\%]]$
	$(buying, buyinglow \rightarrow buyinglow) \wedge (maint, maintvhigh \rightarrow maintvhigh) \wedge (persons, persons2 \rightarrow persons4) \wedge (safety, safetyhigh \rightarrow safetyhigh) \Longrightarrow (Class, unacc \rightarrow acc)[Support : 232\&Confidence : 100\%]]$

6.4 SARGS Experiment - in Apache Spark

We used datasets: Car Evaluation dataset and Mammographic-mass dataset shown in [95] to test our system and compare the results with the Hadoop system. These datasets are publicly available Machine Learning Repository generated by Department of Information and Computer Science of the University of California, Irvine. Since these datasets are relatively small for the distributed frameworks, we replicated multiple times to execute the proposed system in a distributed environment. Also, we used NPS (Net Promoter Score) [20] dataset to test our system on a real time data.

Table 6.6. gives details about the datasets such as number of instances, attribute names, decision attribute values and data size.

Table 6.6: Properties of the datasets for SARGS

Property	**Car Evaluation Data**	**Mammographic Mass Data**	**Business Data**
# of instances	1728	961	2236
Attributes	7 attributes -Buying -Maintenance -Doors -Persons -Luggage Boot -Safety -Class	6 attributes -BI-RADS -Patient's age -Shape -Margin -Density -Severity	17 attributes
Decision attribute values	Class (unacc, acc, good, vgood)	Severity (0 - benign, 1-malignant)	Promoter Status (Detractor, Passive, Promoter)
# of instances / decision value	unacc - 1210 acc - 384 good - 69 vgood - 65	0 - 516 1 - 445	Detractor-112 Passive - 238 Promoter-1870
Replication Factor	1024	2048	-
# of instances after replication	1,769,472	1,968,128	-
Original data size	52 KB	16 KB	261 KB
Data size after replication	52 MB	26 MB	-

The Car Evaluation dataset [95] is to evaluate cars about their goodness and acceptability based on their buying frequency, maintenance cost, price and other characteristics related to cars such as safety measure, number of persons it can carry and luggage boot size. Mammography is the most effective method for breast cancer

screening. The Mammographic mass dataset [95] is to measure the severity of breast cancer based on BI-RADS (measures how severe the breast cancer), patient's age, shape and density of the cancer. Out of these attributes, we set some parameters which are given to the algorithms. Table 6.7 shows the parameters that we set for the selected datasets.

Table 6.7: Parameters used for Action Rule discovery used for both MR-random Forest and SARGS

Property	**Car Evaluation Data**	**Mammographic Mass Data**	**Business Data**
Stable attributes	Buying, Maintenance, Doors	Shape, Age	Client Name, Division, Survey Type, Channel Type
Required decision action	(Class) $unacc \rightarrow acc$	(Severity) $1 \rightarrow 0$	Promoter Status $Detractor \rightarrow Promoter$
Minimum Support and Confidence	2000, 60%	500, 60%	2, 60%

The NPS (Net Promoter Score) [20] dataset is collected customer feedback data related to heavy equipment repair. The entire dataset consists of 38 companies, located in different sites across the whole United States as well as several parts of Canada. Overall, there are about 340,000 customers surveyed in the database over time span of 2011-2015. Customers were randomly selected to answer a questionnaire which was specifically designed to collect information relevant to NPS (structured into so-called "benchmarks"). All the responses from customers were saved into database with each question (benchmark) as one feature in the dataset. Benchmarks include numerical scores (0-10) on different aspects of service: e.g. if job done correctly, are you satisfied with the job, likelihood to refer, etc. The dataset also contains customer details (name, contact, etc.) and service details (company, invoice, type of equipment

repaired, etc.). The decision attribute in the dataset is *PromoterStatus* which labels each customer as either $promoter, passive$ or $detractor$. The decision problem here is to improve customer satisfaction / loyalty as measured by Net Promoter Score. The goal of applying Action Rules to solve the problem is to find minimal sets of actions so that to "reclassify" customer from "Detractor" to "Promoter" and the same improve NPS.

We use the University of North Carolina at Charlotte Research Cluster to evaluate the algorithms on the chosen datasets. This is a Hadoop cluster comprising of 72 nodes. For testing purpose, we also run Hadoop system on the selected datasets and the results are evaluated with the current system. Also, we tested the system both in single node and 4 nodes and many evaluations are given below.

Table 6.8 shows sampled Action Rules extracted from all selected datasets. ARC, ARM and ARN represents Action Rules extracted from Car evaluation, Mammographic mass and NPS datasets respectively. First, we give sample Action Rules that obtained using the Hadoop system [46] and then we give details about Action Rules extracted from all three datasets.

Following Action Rules are generated by the Hadoop system for the Car Evaluation dataset:

- $(maint, high \rightarrow high) \wedge (safety, low \rightarrow high) \wedge (buying, high \rightarrow high) \wedge (persons, \rightarrow 4) \longrightarrow (class, unacc \rightarrow acc)$
- $(maint, low \rightarrow low) \wedge (persons, 2 \rightarrow 4) \wedge (buying, vhigh \rightarrow vhigh) \wedge (safety, \rightarrow high) \longrightarrow (class, unacc \rightarrow acc)$

Since the Hadoop system uses ARoGS algorithm, some of the action terms in the above Action Rules remain meaningless. For example, the action terms $(persons, \rightarrow 4)$ and $(safety, \rightarrow high)$, means that the attributes $persons$ and $safety$ can from any

Table 6.8: SARGS output samples - Action Rules

Car Dataset

1. AR_{C1} : $(buying = med) \wedge (doors = 3) \wedge (maint = med) \wedge (persons, 2 \rightarrow more) \wedge (safety, high \rightarrow med) \longrightarrow (class, unacc \rightarrow acc)[Support : 6000, OldConfidence : 100\%, NewConfidence : 100\%]$

2. AR_{C2} : $(buying = high) \wedge (doors = 3) \wedge (maint = low) \wedge (persons, 4 \rightarrow more) \wedge (safety, low \rightarrow high) \longrightarrow (class, unacc \rightarrow acc)[Support : 6000, OldConfidence : 100\%, NewConfidence : 100\%]$

Mammographic Mass Dataset

1. AR_{M1} : $(BI-RADS, 55 \rightarrow 2) \longrightarrow (Severity, 1 \rightarrow 0)[Support : 14700, OldConfidence : 100\%, NewConfidence : 100\%]$

2. AR_{M4} : $(BI-RADS, 5 \rightarrow 4) \wedge (Margin, 5 \rightarrow 2) \wedge (Shape = 3) \longrightarrow (Severity, 1 \rightarrow 0)[Support : 2100, OldConfidence : 100\%, NewConfidence : 100\%]$

NPS Dataset

1. AR_{N3} : $(BM : DealerCommunication, Low \rightarrow VeryHigh) \wedge (BM : EaseofUseforOnlineStore, Low \rightarrow High) \longrightarrow (PromoterStatus, Detractor \rightarrow Promoter)[Support : 19, OldConfidence : 100\%, NewConfidence : 100\%]$

2. AR_{N4} : $(BM : DealerCommunication, Low \rightarrow High) \wedge (BM : OverallSatisfaction, Medium \rightarrow VeryHigh) \wedge (BM : PartsOrderAccuracy, Low \rightarrow VeryHigh) \wedge (BM : PartsAvailability, Medium \rightarrow High) \wedge (ClientName =' Client2') \wedge (Division =' Tractor\&Equipment') \longrightarrow (PromoterStatus, Detractor \rightarrow Promoter)[Support : 3, OldConfidence : 100\%, NewConfidence : 100\%]$

values to $'4'$ and $'high'$ respectively. Since this kind of Action Rules are incomplete, they provide only limited knowledge to the users.

To overcome this scenario, we proposed *SARGS* algorithm to extract complete

Action Rules. Consider AR_{C1}, AR_{M4} and AR_{N4} from Table 6.8.

$AR_{C1} : (buying = med) \wedge (doors = 3) \wedge (maint = med) \wedge (persons, 2 \rightarrow more) \wedge (safety, high \rightarrow med) \longrightarrow (class, unacc \rightarrow acc)[Support : 6000, Confidence : 100\%]$

This action rule defines that when a $'BuyingCost'$ is $'Medium'$, $'DoorCount'$ is $'3'$, $'MaintenanceCost'$ is $'Medium'$, if $'SeatingCapacity'$ increases from $'2'$ to $'morethan5'$ and if $'Safetymeasure$" decreases from $'high'$ to $'medium'$, some of the unacceptable cars can become acceptable with support of 6000 and confidence of 100%

$AR_{M4} : (BI-RADS, 5 \rightarrow 4) \wedge (Margin, 5 \rightarrow 2) \wedge (Shape = 3) \longrightarrow (Severity, 1 \rightarrow 0)[Support : 2100, OldConfidence : 100\%, NewConfidence : 100\%]$

This action rule means that when $'BI - RADSmeasure'$ decreases from $'5'$ to $'4'$, $'Marginvalue'$ decreases from $'5'$ to $'2'$ and if the $'shapeofthetumor'$ is $'3'$, severity of the cancer can be reduced.

$AR_{N4} : (BM : DealerCommunication, Low \rightarrow High) \wedge$
$(BM : OverallSatisfaction, Medium \rightarrow VeryHigh) \wedge$
$(BM : PartsOrderAccuracy, Low \rightarrow VeryHigh) \wedge$
$(BM : PartsAvailability, Medium \rightarrow High) \wedge (ClientName =' Client2') \wedge$
$(Division =' Tractor\&Equipment') \longrightarrow (PromoterStatus, Detractor \rightarrow Promoter)$
$[Support : 3, Confidence : 100\%]$

This action rule defines that when benchmarks (BM): $'DealerCommunication'$ increases from $'Low'$ to $'High'$, $'OverallSatisfaction'$ increases to $'VeryHigh'$, 'Parts Order Accuracy' increases to $'VeryHigh'$ and $'PartsAvailability'$ increases to $'High'$ and when the $'Client'$ is $'Client2'$ and when the $'Division'$ is $'Tractor\&Equipment'$, the $'PromoterStatus'$ can change from $'Detractor'$ to $'Promoter'$ with support of 3 and confidence of 100%. By making changes provided in this action rule, some of the

customer who are not supporting (Detractors) currently can convert into Promoters.

Thus, by using our proposed *SARGS* algorithm all action terms are complete. So, the knowledge given by any action rule formed by such complete action terms from our system is also completely meaningful.

Table 6.9 shows the number unique Action Rules produced by the two systems for the selected datasets. Since Car evaluation and Mammographic mass datasets are relatively small, we tested our data sampling method (manually distributing the data partitions to nodes) only NPS data. When Spark distribute the data among the nodes, our system could not able to find some of the valuable Action Rules and it extract only 1200 rules. This count continues to decrease when data size increases and Spark runs our system in more partitions. But with manual sampling method defined in section III.G, our system extracts more 1359 Action Rules. For Car evaluation and Mammographic mass datasets, with our *SARGS* algorithm, equal number of Action Rules are extracted from the system unlike MR-Random Forest algorithm which skips most of the rules while running in multiple nodes.

Table 6.10 shows running time of our systems in both single node and multiple nodes. It shows that our Spark system has the ability to complete the entire processing within seconds for which the Hadoop system took minutes to complete. that our present system works way faster than our previous Hadoop system. This improved running time efficiency is due to the Spark framework [19] and its ability to perform in-memory computations.

6.5 Data Distribution - Class Attribute Sampling Experiment

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

The Car Evaluation and Mammography are obtained from the Machine Learning

Table 6.9: Action Rules Count Evaluation of MR-Random Forest and SARGS on the datasets

Dataset	**# of nodes**	**MRRandom on Hadoop (Action Rules Count)**	**SARGS on Spark (Action Rules count)**
Business Data	1	-	1265
	2 (default data distribution)	-	1200
	2 (manual data distribution)	-	1359
Car Evaluation	1	950	251
	4	230	251
Mammographic Mass Data	1	673	319
	4	145	319

Table 6.10: Runtimes Evaluation of MR-Random Forest and SARGS on the datasets

Dataset	**# of nodes**	**Hadoop (minutes)**	**Spark (minutes)**
Business Data	1	-	6.2
	2	-	5.5
Car Evaluation	1	2	1.2
	4	1.4	0.95
Mammographic Mass Data	1	1.55	0.66
	4	0.8	0.4

repository of the Department of Information and Computer Science of the University of California, Irvine [95]. The Car Evaluation Data consists of records describing a car's goodness and acceptability based on features such as buying frequency, maintenance cost, safety measure, seating capacity and luggage boot size. Mammographic is the most effective method for screening breast cancer. The Mammographic Mass data contains records that measure severity of the cancer based on patient's age, can-

cer shape, cancer density and BI-RADS(a test score to denote how severe the cancer is). Since these datasets are relatively small in size, in order to test them for scalability with the proposed distributed processing algorithms, we replicate their data rows 1024 and 2056 times respectively for CarEvaluation and MammographicMass datasets, in order to increase data size. We test our algorithm on both replicated and non-replicated data for CarEvaluation and Mammographic datasets.

We also test with the city of Charlotte North Carolina BusinesWise data , which is donated by the Charlotte Chamber of Commerce. This data collects details of over 20,000 business companies in Mecklenburg county, North Carolina. The data includes their City, StartYear, Sector, Specialization of the company in a selected sector, SiteType, Employees count at the site, Total employees in the company including all branches, Site building type, Total sites and Estimated Sales. From this data, our focus is how to increase a company's estimated sales from $<$2 million US dollars to the range between 3 million and 10 million US dollars. Further Table 6.11 gives a broad picture of the datasets, , like number of instances, replication factor and data size, that we used to test our algorithm.

Table 6.12 show parameters that we set for each dataset to collect Action Rules. For the *Car Evaluation* data, we choose *Class* attribute as a decision attribute and we collect Action Rules to help the car company to change the car from *Unacceptable* state to *Acceptable* state. For the *Mammographic Mass* data, we choose *Cancer Severity* as a decision attribute and we collect Action Rules to reduce the severity from *Malignant* to *Benign*. For the *Business* data, we choose *Estimated Sales* as a decision attribute and we collect Action Rules to increase the Estimated Sales of a company from $< \$3M$ to $\$3M - \$10M$. For our vertical data distribution method evaluations, we split the Business data only because of the small number of attributes in Car Evaluation data and Mammographic Mass data.

In Table 6.13, we give number of Action Rules extracted from our different method-

Table 6.11: Properties of datasets for Data Distribution Experiments

Property	**Car Eval. Data**	**Mammo. Mass Data**	**Business Data**
Attributes	7 attributes -Buying -Maintenance -Doors -Persons -Luggage Boot -Safety -Class	6 attributes -BI-RADS -Patient's age -Shape -Margin -Density -Severity	17 attributes including -City -Sector -Site Type -Building Type -Estimated Sales -Total Employees
Decision attribute values	Class (unacc, acc, good, vgood)	Severity (0 - benign, 1- malignant)	Est. Sales $< \$2M$,3-10M, 10-25M, 25-50M, 50-100M,100-500M, $> 500M$
# of instances / decision value	unacc - 1210 acc - 384 good - 69 vgood - 65	0 - 516 1 - 445	$< \$2M$ - 12503 \$2-\$10M - 1927 \$10-\$25M - 393 \$25-\$50M - 130 \$50-\$100M - 69 \$100-\$500M - 57 $> \$500M$ - 50
Replication Factor	1024	2048	-
# of instances after replication	1,769,472	1,968,128	-
Original data size	52KB	16KB	5.5MB
Data size after replication	52MB	26MB	-

ologies: MR-Random Forest Algorithm, SARGS with default data partitioning and SARGS with class distribution methods which are based on *LERS* and *ARAS* and we validate our methods with the traditional non-parallel algorithm.

In Table 6.14, we give runtimes of our MR-Random Forest and SARGS methods and compare them with the runtime of traditional Action Rule extraction algorithm. Note that our SARGS implementation with Class Distribution technique take some extra time for data partitioning task. But time taken for this step gradually reduces

Table 6.12: User defined Parameters used in all Action Rule discovery algorithms

Property	**Car Eval. Data**	**Mammo. Mass Data**	**Business Data**
Stable attributes	Maintenance, Buying Price, Doors	Age, Shape	Start Year
Required decision action	(Class) $unacc \rightarrow acc$	(Severity) $1 \rightarrow 0$	(Est. Sales $\$2M - \$10M \rightarrow \$10M - \$25M$
Minimum Support α and Confidence β	2048, 70%	4096, 70%	10, 70%

Table 6.13: Performance of Action Rule extraction algorithms in terms of number of rules generated

Data	**Non-Parallel Algorithm**	**MR-Random Forest**	**SARGS - Default Data Distribution**	**SARGS - Class Distribution Algorithm**
Car Evaluation Data	64	49	53	53
Mamm. Mass Data	393	125	165	165
Business Data	N/A	N/A	2048	3100

when the data size grows bigger. Avoiding that, our distributed versions of algorithms runs much more efficiently compared to the traditional non-parallel algorithm.

6.6 Data Distribution - Vertical Split Experiment

In Table 6.15, we give analysis of Association Action Rules extraction methods using MapReduce(MR-Apriori) and Spark(Vertical Data Distribution) methods. We validate our methods with the traditional Association Action Rules extraction method. From this table, it is noticeable that we can reduce Action Rule loss while executing them in distributed computing frameworks using our vertical data partitioning technique compared to our random horizontal data partitioning in MapReduce.

Table 6.14: Time taken by Action Rule extraction algorithms to extract Action Rules

Data	**Non-Parallel Algorithm**	**MR-Random Forest Algorithm**	**SARGS - Default Data Distribution**	**SARGS - Class Distribution**
Car Evaluation Data	34secs	25secs	17secs	30secs
Mamm. Mass Data	45secs	36secs	23secs	32secs
Business Data	$> 5hr$	N/A	$\sim 3hr$	$\sim 3hr$

Table 6.15: Performance of Association Action Rules algorithms in terms of number of rules generated

Data	**Non-Parallel Algorithm**	**MR-Apriori Forest Algorithm**	**Vertical Data Distribution Algorithm**
Car Evaluation Data	3512	2478	3496
Mamm. Mass Data	5802	4892	5756
Business Data	~ 210000	N/A	~ 210000

In Table 6.16, we give runtimes of our Association Action Rules techniques and compare them with non-parallel method. It is worth mentioning here that this Association Action Rules technique is more complex than other Action Rules extraction methods because this method extracts all possible rules from the Decision Table. We found that our distributed algorithms works much faster than the traditional method. Within our distributed methods, it is more interesting to note that our vertical data distribution approach works much faster than our random and horizontal data distribution method.

Along with the above datasets, we used Net Promoter Score(NPS) dataset also to evaluate this vertical data partitioning method. For this data, we just make comparison between the traditional version of the Association Action Rules algorithm and our Distributed version of Association Action Rules algorithm using Spark framework.

Table 6.16: Time taken by Association Action Rules extraction algorithms to extract Action Rules

Data	**Non-Parallel Algorithm**	**MR Apriori Algorithm**	**Vertical Data Distribution method**
Car Evaluation Data	72mins	11mins	104secs
Mamm. Mass Data	40mins	9.3mins	82secs
Business Data	$> 20hrs$	N/A	$2.3hr$

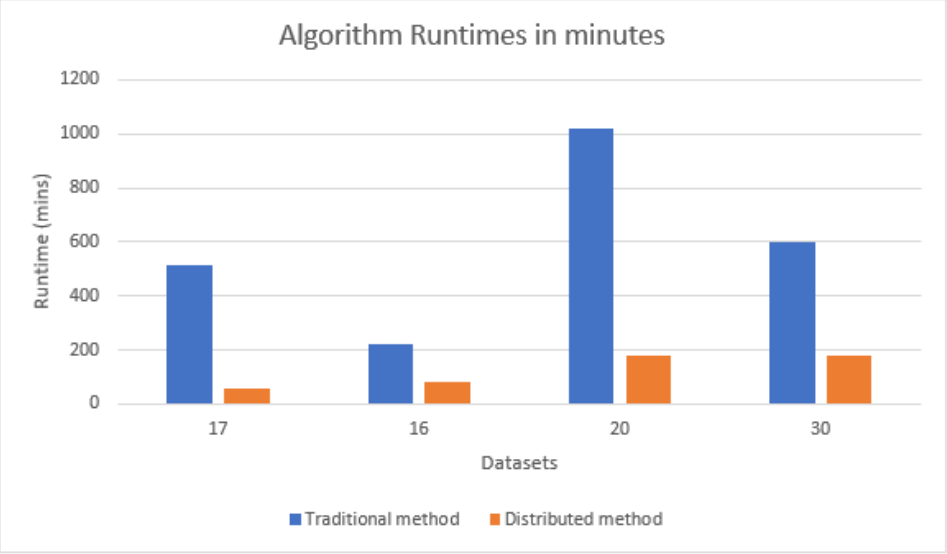


Figure 6.9: Runtimes of Association Action Rules extraction algorithms for NPS datasets in Traditional vs Spark Environments

From Figure 6.9, we can note that our algorithms with vertical data distribution outperforms the traditional version of the algorithm. We noted some interesting patterns during this evaluations. For datasets like *Company_16* and *Company_30* which contains more number of instances than datasets *Company_17* and *Company_20* respectively, the traditional algorithms outputs actionable recommendations in a short time. This is due to many empty values in those datasets. Our Spark version of the algorithm takes same time to produce results due to data separation and combining results overload in its process.

In Table 6.17, we compare coverage of traditional and Spark versions of algorithms.

It is notable that our Spark version loses some coverage due to minor errors in data distribution in some datasets.

Table 6.17: Comparison of coverage of Association Action Rules extraction algorithms for NPS datasets

Data	**Traditional algorithm**	**Spark algorithm**
Company_17 - 547 rows	77.3%	77.3%
Company_16 - 2078 rows	75%	76.9%
Company_20 - 2590 rows	80.5%	81.8%
Company_30 - 3335 rows	79.8%	79.8%

6.7 Vertical Data Split with Information Granules

To test the proposed Vertical Data Split method with Information Granules, we use three datasets: *Car Evaluation* data [95], *Mammographic Mass* data [95], and the Net Promoter Score dataset data [96].

Table 6.18 show parameters that we set for each dataset to collect Action Rules. For the *Car Evaluation* data, we choose *Class* attribute as a decision attribute and we collect Action Rules to help the car company to change the car from *Unacceptable* state to *Acceptable* state. For the *Mammographic Mass* data, we choose *Cancer Severity* as a decision attribute and we collect Action Rules to reduce the severity from *Malignant* to *Benign*.

We show the Action Rules extracted using our methods in two tables. In Table 6.19, we give Action Rules extracted from Car Evaluation and Mammographic Mass datasets and in Table 6.20, we give Action Rules extracted from 4 NPS datasets. These Action Rules provide actionable recommendations to users who wants to achieve the desired decision action. For example, from Table 6.19, when a user wants to

Table 6.18: Parameters used in all Action Rule discovery algorithms

Property	**Car Evaluation Data**	**Mamm. Mass Data**	**NPS Data**
Stable attributes	Maintenance, Buying Price, Doors	Age	Survey name Survey type Division Channel type Client name
Required decision action	(Class) $unacc \rightarrow acc$	(Severity) $1 \rightarrow 0$	Promoter Status *Detractor* $\rightarrow$ *Promoter*
Minimum Support α and Confidence β	2048, 70%	4096, 70%	2, 80%

achieve the action $Class, unacc \rightarrow acc$, our system give actionable recommendations to achieve that goal. Action Rules AR_{C1} and AR_{C2} in Table 6.19 are example recommendations given by our system for the appropriate parameters given in Table 6.18. For example, Action Rule AR_{C1} recommends if the car company maintains *Buying cost* to *medium* and *Maintenance Cost* to *Very high* and decrease the *Seating capacity* from *More than 4* to *4* and increase *Safety measures* from *medium* to *high* with support of 1107 and minimum confidence of 74%

Similarly, we give example Action Rules for all NPS datasets: *Company_16, Company_16_31, Company_17 and Company_30* in Table 6.20. In all cases, we use parameters given in Table 6.18. For example, consider AR_{N2} which recommends that if the company can improve user's ratings on *Completion of repair correctly* from *5 points* to *10 pints* and improve user's ratings on *Technician communication* from *3 points* to *9 points*, the company can convert some *Detractors* to *Promoters* with support of 2.0 and confidence of 90.0%.

In Figure 6.10, we give run time analysis of Association Action rules extraction methods implemented in non-parallel method and our technique of splitting the data

Table 6.19: Action Rules of Car Evaluation and Mammographic Mass datasets

Car Evaluation Data
1. $AR_{C1} : (buying = med) \wedge (maint = vhigh) \wedge (persons, more \rightarrow 4) \wedge (safety, med \rightarrow high) \Longrightarrow (class, unacc \rightarrow acc)[Support : 1107, OldConfidence : 74\%, NewConfidence : 100\%, Utility : 74\%$
2. $AR_{C2} : (buying = med) \wedge (maint = vhigh) \wedge (persons, more \rightarrow 4) \wedge (safety, med \rightarrow high) \Longrightarrow (class, unacc \rightarrow acc)[Support : 1353, OldConfidence : 85\%, NewConfidence : 100\%, Utility : 85\%$
Mammographic Mass Data
1. $AR_{M1} : (BIRADS, 6 \rightarrow 2) \wedge (Density, 4 \rightarrow 3) \Longrightarrow (Severity, 1 \rightarrow 0)[Support : 12288, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%]$
2. $AR_{M2} : (Age = 42) \wedge (Density, 1 \rightarrow 3) \wedge (Shape = 1) \Longrightarrow (Severity, 1 \rightarrow 0)[Support : 10240, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%]$

by attributes using information granules and extracting Association Action rules in parallel using Apache Spark framework. As mentioned earlier, since Association Action rules methodology is a complex algorithm, we can see from the Figure 6.10 that non-parallel method takes long time to give actionable recommendations. On the other hand, our method takes only fraction of minutes. The speed increases when the data set size increases. For example, for NPS data: company 30, which is the largest in our datasets, the non-parallel version takes around an 60 minutes to produce results, while our method takes 3 minutes.

In Table 6.21, we compare number of Action Rules extracted by our parallel and non-parallel algorithms. Due to the data partitioning step involved in our method, we lose some Action Rules. In Table 6.22, we give Coverage measure of Action

Table 6.20: Action Rules of NPS datasets: 16, 16_31, 17, 30

Company_16

1. AR_{N1} : $(ChannelType = Ag) \wedge (Benchmark : Service - TechnicianCommunication, 1 \rightarrow 10) \Longrightarrow (PromoterStatus, Detractor \rightarrow Promoter)[Support : 2.0, OldConfidence : 87.5\%, NewConfidence : 100.0\%, Utility : 87.5\%]$

2. AR_{N2} : $(Benchmark : Service - RepairCompletedCorrectly, 5 \rightarrow 10) \wedge (Benchmark : Service - TechnicianCommunication, 3 \rightarrow 9) \Longrightarrow (PromoterStatus, Detractor \rightarrow Promoter)[Support : 2.0, OldConfidence : 90.0\%, NewConfidence : 90.0\%, Utility : 100.0\%]$

Company_16_31

1. AR_{N3} : $(BenchmarkPartsOrderAccuracy, 3 \rightarrow 10) \wedge (ClientName = HOLTCAT) \rightarrow (Division = Parts) \wedge (BenchmarkPartsTimeitTooktoPlaceOrder, 4 \rightarrow 9) \Longrightarrow (PromoterStatus, Detractor \rightarrow Promoter)[Support : 2.0, OldConfidence : 80.0\%, NewConfidence : 100.0\%, Utility : 80.0\%]$

2. AR_{N4} : $(BenchmarkPartsHowOrdersArePlaced, 2 \rightarrow 4) \wedge (BenchmarkPartsOrderAccuracy, 6 \rightarrow 10) \wedge (ClientName = HOLTCAT) \wedge (Division = Parts) \wedge (BenchmarkPartsKnowledgeofPersonnel, 5 \rightarrow 10) \Longrightarrow (PromoterStatus, Detractor \rightarrow Promoter)[Support : 2.0, OldConfidence : 100.0\%, NewConfidence : 100.0\%, Utility : 100.0\%]$

Rules extracted for each dataset and compare them with Coverage of Action Rules extracted using non-parallel methods. In 50% of our experiments we are able to achieve the same coverage as Action Rules extracted using non-parallel approach. From this table, we can assure that our method of extracting Action Rules using the distributed computing framework like Spark [19] can produce results in a faster runtime by losing only a limited knowledge.

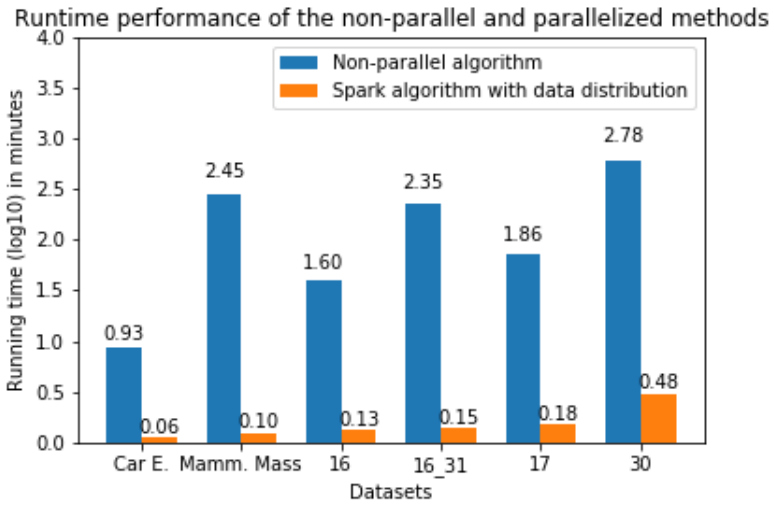


Figure 6.10: Speed Performance of Non-parallel and Parallel algorithms of Association Action Rules extraction

Table 6.21: Performance, in terms of number of resulting Action Rules, using parallel and non-parallel versions of Association Action Rules extraction methods for all datasets

Dataset	**Non-parallel algorithm**	**Parallel algorithm**
Car Evaluation data	3500	3496
Mammographic Mass data	5790	5756
NPS data: Company_16	900	798
NPS data: Company_16_31	83643	83000
NPS data: Company_17	37256	37000
NPS data: Company_30	184000	182000

Table 6.22: Performance of algorithms for all datasets in terms of a Coverage measure

Dataset	**Non-parallel algorithm**	**Parallel algorithm - Random distribution**	**Parallel algorithm - Granule distribution**
Car Evaluation data	99.5%	99%	99%
Mammographic Mass data	94%	92.53%	92.53%
NPS data: Company_16	89.2%	86.9%	86.9%
NPS data: Company_16_31	73%	70.77%	70.77%
NPS data: Company_17	72.7%	72.7%	72.7%
NPS data: Company_30	75.5%	75.5%	75.5%

6.8 Semantic Data Partitioning Results

In this section, we show the robustness of our proposed method using the HCUP data for the State of Florida [97] and a massive replicated data of the Car Evaluation data. In these experiments, we aim to show how our proposed method perform not only with the improved execution time efficiency but also with improved resource utilization and load balancing.

Since we followed semantic data distribution, we split the data by their diagnosis codes first. We set these partitioned data as a source of input for algorithms. For vertical data distribution algorithms, we split these data again by their attributes. Since there are 208 diagnosis codes found in the data(208 different datasets), we give only the results of few diagnosis due to space constraints. Finally we give results of the cumulative data. With all diagnosis we show the execution time of all algorithms. For the cluster based algorithms, we also give the amount of loads per process for

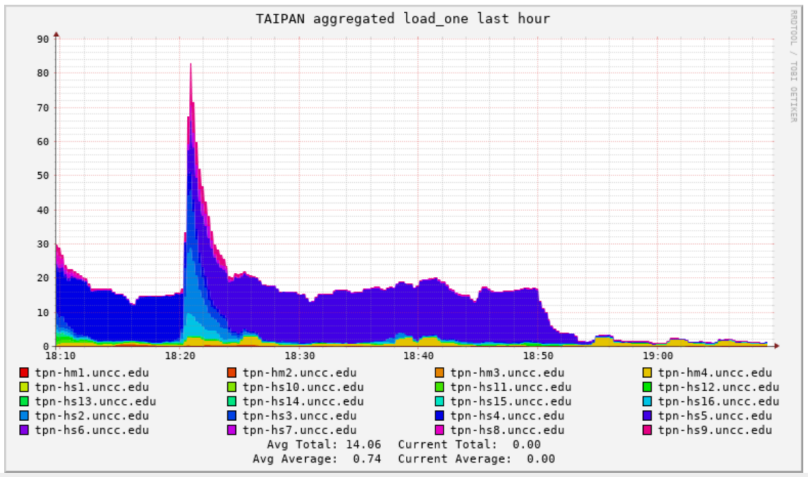
Table 6.23: Execution time of algorithms for the HCUP data

Dataset	**Non-parallel algorithm**	**SARGS**	**Vertical Data Distribution**	**Semantic Data Distribution**
670(Mental Health Disorders)	>2 days	2.14 hours	49 mins	13.7 mins
250(Nausea & Vomiting)	>2.5 days	2.62 hours	1.14 hours	17 mins
654(Developmental disorders)	>2days	1.8 hours	35 mins	11.5 mins
233(Intracranial injury)	>2days	2.3 hours	57 mins	13.8 mins
236(Open wounds)	>2days	2 hours	46 mins	8.6 mins

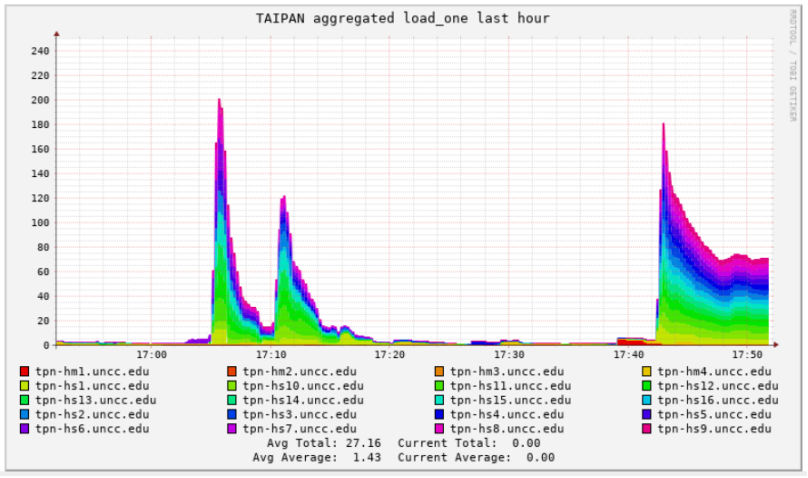
each algorithm.

In Table 6.23, we give the execution time of all our proposed algorithms. We can note that for big datasets like the HCUP data, the non-parallel version of the action rule extraction method takes very long time to extract action rules. Whereas, the proposed cloud based performs much faster than the non-parallel methods. Particularly, our latest paralellized vertical data distribution algorithm - the semantic data distribution - achieves better execution time compared to our other cloud based counterparts.

In Figure 6.11, we give total node usage in the cluster by simple vertical data distribution and semantic data distribution algorithms for the diagnosis code *250*. It is notalble from this figure that the simple vertical data distribution takes more than *1 hour* to complete extracting all actionable patterns, whereas the semantic data distribution taken only around *17 minutes* for extracting the recommendations. Also we can see that in the vertical data distribution method from Figure 6.11a, one or two nodes execute most of the time and in our semantic data distribution method from Figure 6.11b utilizes much more parallalization in the cluster.



(a) Vertical Data Distribution cluster usage

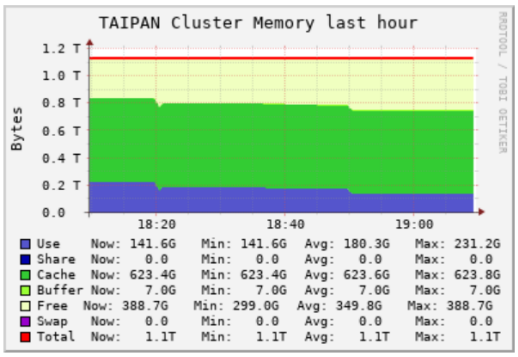


(b) Semantic Data Distribution cluster usage

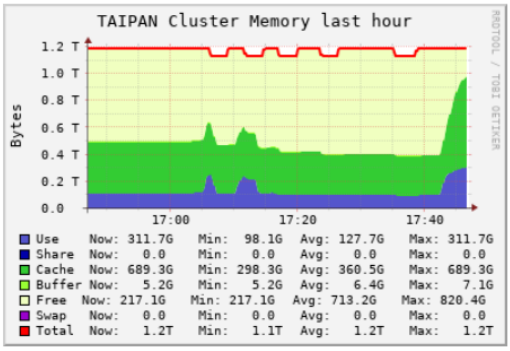
Figure 6.11: Total cluster usage by simple vertical data distribution and semantic data distribution

We also give the cluster memory usage in Figure 6.12. We can note from Figure 6.12a that the vertical data distribution method starts with loading large quantity of data in the memory and slowly it reduces as the algorithm starts extracting the recommendations. However, the semantic data distribution in Figure 6.12b occupies only limited quantity of data in the memory while the algorithm progress in extracting actionable patterns.

In Table 6.44, we give action rules of diagnoses given in Table 6.23. We consider these actionable patterns as recommendations to hospitals in such a way that for a



(a) Vertical Data Distribution cluster memory

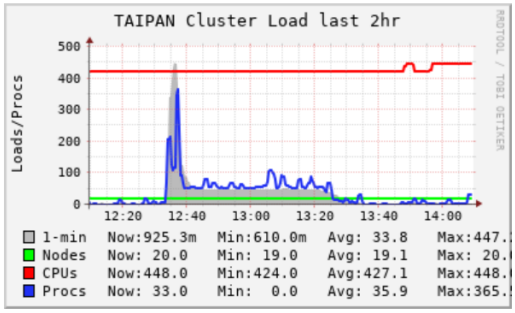


(b) Semantic Data Distribution cluster memory

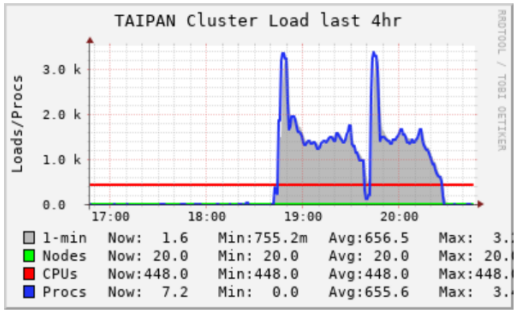
Figure 6.12: Total cluster memory occupied by simple vertical data distribution and semantic data distribution

given disease, if the hospital provides treatment or care for recommended diseases, hospitals can potentially reduce the number of hospital readmissions. For example with disease code *250*, if hospitals give treatment for disease codes *21* (*Bone cancer*) and *251* (*Abdomen pain*), hospitals can reduce readmission by 50%. The support of *143* shows that the framework identifies 143 entries in the data to acquire this change.

6.9 Experiments on Massive Dataset



(a) SARGS job load



(b) Vertical Data Distribution job load

Figure 6.13: Load of algorithms for the massive dataset

We also conduct experiments on the applicability of our proposed algorithms with massive datasets with data size of half terabytes and with approximately 1 billion

Table 6.24: Sample Action Rules from the HCUP data for selected diagnosis

670(Mental Health Disorders)

250(Nausea & Vomiting)

1. $H250_{AR1}$: $(DX, 21(\text{Bone Cancer}) \rightarrow 251(\text{Abdominal pain})) \Rightarrow (Readmission, 1- > 0)[Support : 143.0, OldConfidence : 49.47\%, NewConfidence : 58.68\%]$

2. $H250_{AR2}$: $(DX, 21(\text{Bone Cancer}) \rightarrow 234(\text{Internal injury})) \Rightarrow (Readmission, 1 \rightarrow 0)[Support : 38.0, OldConfidence : 52.92\%, NewConfidence : 60.52\%]$

654(Developmental disorders)

1. $H654_{AR1}$: $(DX, 238(\text{Surgical procedure complication}) \rightarrow 11(\text{Neck/Head Cancer})) \Rightarrow (Readmission, 1 \rightarrow 0)[Support : 49.0, OldConfidence : 44\%, NewConfidence : 51.43\%]$

2. $H654_{AR2}$: $(DX, 45(\text{Radiotherapy}) \rightarrow 100(\text{Myocardial infarction})) \Rightarrow (Readmission, 1 \rightarrow 0)[Support : 44.0, OldConfidence : -51\%, NewConfidence : 63.72\%]$

233(Intracranial injury)

1. $H233_{AR1}$: $(DX, 228(\text{Skull fracture}) \rightarrow 52(\text{Nutritional deficiency})) \Rightarrow (Readmission, 1 \rightarrow 0)[Support : 65.0, OldConfidence : 40.85\%, NewCOnfidence : 51.5\%]$

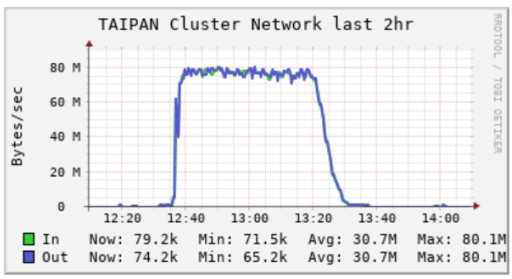
2. $H233_{AR2}$: $(DX, 131(\text{Respiratory failure}) \rightarrow 108(\text{Congestive heart failure})) \Rightarrow (Readmission, 1 \rightarrow 0)[Support : 72.0, OldConfidence : 33.10\%, NewConfidence : 40.24\%]$

236(Open wounds)

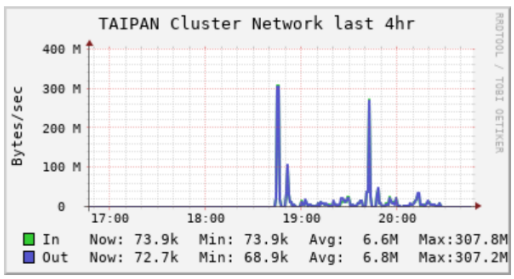
1. $H236_{AR1}$: $(DX, 236(\text{Open wounds}) \rightarrow 197(\text{Skin infection}) \&654(\text{Developmental disorders})) \Rightarrow (Readmission, 1 \rightarrow 0)[Support : 37.0, OldConfidence : 63.18\%, NewConfidence : 78.57\%]$

2. $H236_{AR2}$: $(DX, 236(\text{Open wounds}) \rightarrow 49(\text{Diabetes})) \Rightarrow (Readmission, 1 \rightarrow 0)[Support : 49.0, OldConfidence : 51.55\%, NewConfidence : 67.79\%]$

data records. For testing purposes, we use the car data described in Section 6.1.1 and replicated the data to 0.5 TeraBytes size. We use *MR-Random Forest* and *SARGS* algorithms for our experiments since the data contains very few attributes(6). To provide robustness of our proposed vertical data distribution method, we use the algorithm but keep the number of vertical partitions as 1. We show the evaluation of our methods through the hadoop cluster's status monitoring system. Figure 6.13 shows the runtime taken by our methods to extract actionable patterns and overall load taken by the proposed methods. Figure 6.14 gives the network activity through out the job. It is evident from these figures that the proposed SARGS algorithm executes much faster(2x times) than that of the Association Action Rules method. We claim that this result is because of very few attributes(6) in the dataset. Moreover, since the data is not split into partitions for Association Action Rules method, it executes longer than expected execution time. It is also notable from Figure 6.13 that SARGS method results only limited load to the cluster. This is because of the design of SARGS method that marks certain rules and extracts Action Rules from the marked rules. On the other hand, with Association Action Rules, since the algorithm functions on the complete input the load also increases accordingly.



(a) SARGS job network activity



(b) Vertical Data Distribution job network activity

Figure 6.14: Network activity of algorithms for the massive dataset

In Figure 6.14, we compare the overall network activity (data transfers during the job execution) for each method. Our association action rules method causes only

limited data transfers between executors during the execution. Even though, the quantity of data transfers is high, such transfers complete very quickly unlike the SARGS method which keeps transferring data constantly over time. In addition, we are able to extract **985** rules from the vertical data partitioning method (which extracts association action rules), whereas the SARGS method extracts only **52** rules from the given massive data with the current parameters.

6.10 Action Rules of Lowest Cost - Correlation Matrix Experiment

To evaluate our approach in a distributed environment we used Car Evaluation dataset and Mammographic Mass dataset [95] from the University of California, Irvine's Machine Learning Repository maintained by the Department of Information and Computer Science. We compare the results with the efficiency to generate lowest cost Action Rules using the existing algorithm. As our approach aims in adapting the algorithm that computes the lowest cost of Action Rules to work with bigger datasets, and as the original datasets are relatively small, we replicate the original datasets multiple times to test the performance of our proposed approach in a distributed environment. Table 2 provides all details about both the datasets such as number of instances, replication factor, attributes, and decision attribute values.

Table 6.27 shows some of the sample Action Rules, Action Rules Cost and Lowest Cost Action Rules extracted for Car dataset. From *Action Rules* section in the Table 6.27, it is notable that our distributed Action Rules extraction algorithm produces more specific Action Rules that are capable of delivering complete knowledge to the user. Consider the rules AR_1, ARC_1 and $LCAR_1$ from Table 6.27.

AR_1 is the action rule that describes when '*SeatingCapacity*' is '*4*' , if '*BuyingCost*' decreases from '*high*' to '*medium*' , if '*LuggageBootSize*' decreases from '*big*' to '*small*' , if '*MaintenanceCost*' decreases from '*veryhigh*' to '*medium*' and if '*SafetyMeasure*' increases from '*med*' to '*high*' , certain cars marked as '*Unacceptable*' can become '*Acceptable*' with support of 5104 and confidence of 100%.

Table 6.25: Properties of the datasets for finding Action Rules of lowest cost

Property	**Car Evaluation-Data**	**Mammographic Mass Data**
Number of instances	1728	961
Replication Factor	1024	2048
Number of instances after replication	1,769472	1,968,128
Attributes	7 attributes -Buying -Maintenance -Doors -Persons -Luggage Boot -Safety -Class	6 attributes -BI-RADS -Patient's age -Shape -Margin -Density -Severity
Decision attribute values	Class (unacc, acc, good, vgood)	Severity (0 - benign, 1- malignant)
Original data size	52 KB	16 KB
Data size after replication	52 MB	26 MB

In order to make the suggested changes in AR1, a Meta-Action [35] has to be defined by an expert in the domain. In this case, the car manufacturer determines the Meta-Action in order to make the specified change occur: 'BuyingCost' decreases from 'high' to 'medium'. For example, what can be done do decrease the overall production cost of this make and model car. Meta-Action is an action about the action. In other words, these are high level actions performed by domain experts, which trigger the changes suggested by the Action Rule [35].

Action Rules in ARC section gives Action Rules and their corresponding cost given by the experts. ARC_1 defines the same meaning as AR_1. This Action Rule provides the *Cost* that is 1100 units.

Action Rules in $LCAR$ section gives low cost Action Rules recommendation for all Action Rules in AR and ARC section. For example, the action rule $LCAR_1$ means

Table 6.26: Parameters used for Action Rule discovery

Property	**Car Evaluation Data**	**Mammographic Mass Data**
Stable attributes	Buying, Persons, Doors	Shape, Age
Required decision action	(Class) $unacc \rightarrow acc$	(Severity) $1 \rightarrow 0$
Minimum Support and Confidence	2000, 60%	500, 60%

that when 'SeatingCapacity' is '*4*', '*LuggageBootSize*' decreases from '*big*' to '*small*', if '*MaintenanceCost*' decreases from '*veryhigh*' to '*medium*' and if '*SafetyMeasure*' increases from '*med*' to '*high*', we can obtain the same results but with reduced cost. These changes would trigger the high cost atomic action term $(buying, high \rightarrow med)$. The algorithm does not stop with providing single low cost Action Rule recommendation. For example, for Action Rule ARC_3 with cost of 1200 units, there are two low cost Action Rules recommendations $LCAR_{3.1}$ with cost of 1000 units and $LCAR_{3.2}$ with cost of 800 units. From these recommendations, the user or a company can choose most desired Action Rule that fit their needs.

Table 6.28 shows running time of the algorithm in single machine as well as in distributed data processing environment. It can be inferred from the results that, with the current approach the computational time for both the datasets has drastically improved with Spark environment, where the entire processing has completed within seconds, which otherwise would take several minutes with single machine.

6.11 Lowest Cost Graph - Experiment 1 - in Spark GraphX

For evaluating our finding Action Rules of lowest cost using Action Graphs method, we used Car Evaluation, Mammographic Mass data and the Charlotte Businesswise datasets. In Table 6.29, we give all parameters that we set for discovering Action Rules using SARGS algorithm and minimum cost threshold ϕ that we set to discover low cost Action Rules. With addition to these parameters, we also give cost of atomic

Table 6.27: Sample output results from Car Dataset

Action Rules

1. $AR_1 : (buying, high \rightarrow med) \wedge (lugBoot, big \rightarrow small) \wedge (maint, vhigh \rightarrow med) \wedge (persons = 4) \wedge (safety, med \rightarrow high) \longrightarrow (class, unacc \rightarrow acc)[Support : 5104, OldConfidence : 100\%, NewConfidence : 100\%]$
2. $AR_2 : (buying, med \rightarrow low) \wedge (lugBoot, big \rightarrow small) \wedge (maint, med \rightarrow low) \wedge (persons = 4) \wedge (safety, low \rightarrow med) \longrightarrow (class, unacc \rightarrow acc)[Support : 5104, OldConfidence : 100\%, NewConfidence : 100\%]$
3. $AR_3 : (buying, high \rightarrow med) \wedge (lugBoot, small \rightarrow big) \wedge (maint, med \rightarrow vhigh) \wedge (persons = more) \wedge (safety, low \rightarrow high) \longrightarrow (class, unacc \rightarrow acc)[Support : 5104, OldConfidence : 100\%, NewConfidence : 100\%]$

Action Rules Cost

1. $ARC_1 : (buying, high \rightarrow med) \wedge (lugBoot, big \rightarrow small) \wedge (maint, vhigh \rightarrow med) \wedge (persons = 4) \wedge (safety, med \rightarrow high)[Cost : 1100]$
2. $ARC_2 : (buying, med \rightarrow low) \wedge (lugBoot, big \rightarrow small) \wedge (maint, med \rightarrow low) \wedge (persons = 4) \wedge (safety, low \rightarrow med)[Cost : 1500]$
3. $ARC_3 : (buying, high \rightarrow med) \wedge (lugBoot, small \rightarrow big) \wedge (maint, med \rightarrow vhigh) \wedge (persons = more) \wedge (safety, low \rightarrow high)[Cost : 1400]$

Low Cost Action Rules

1. $LCAR_1 : (lugBoot, big \rightarrow small) \wedge (maint, vhigh \rightarrow med) \wedge (persons = 4) \wedge (safety, med \rightarrow high)[Cost : 800]$
2. $LCAR_2 : (lugBoot, big \rightarrow small) \wedge (maint, med \rightarrow low) \wedge (persons = 4) \wedge (safety, low \rightarrow med)[Cost : 1000]$
3. $LCAR_{3.1} : (lugBoot, small \rightarrow big) \wedge (persons = more) \wedge (safety, low \rightarrow high)[Cost : 1000]$
4. $LCAR_{3.2} : (buying, high \rightarrow med) \wedge (maint, med \rightarrow vhigh) \wedge (persons = more) \wedge (safety, low \rightarrow high)[Cost : 800]$

Table 6.28: Evaluation of lowest cost Action Rules in Single Machine vs Spark Environment

Dataset	**# Records**	**# Action Rules**	**Single Machine Algorithm Running time (min)**	**Spark Algorithm Running time(sec)**
Car Evaluation	1728	40	1.1	4
	2,204,927	415	>15	6.5
Mamm. Mass Data	961	185	0.29(17 sec)	4.5
	1,968,124	443	1.8	6.25

action terms. Usually, the cost of each action term is specified by an expert in the domain. For example, for the Mammography dataset, a medical doctor specifies the cost for the suggested actions. For Car Evaluation data, the car manufacturer specifies the cost for the suggested actions. However, for our experiment purpose, we assign a random cost number to each Action Term. We assign the cost of *0* for ActionTerms which have stable attributes, because the stable attributes cannot be changed. For the Flexible Attributes Action Terms the cost values are between 0 and 1000. We calculate total cost of each Action Rule by adding the cost of all action terms in the rule.

Next, we build an Action Graph using the list of extracted Action Rules as an input. We implement the Action Graph in both non-parallel environment, and in a clustered environment for performance and scalability comparision. The Non-parallel version is implemented in Java. The Apache Spark [19] using the Spark GraphX library. We use Scala programming language. We test the system on a Spark cluster running over Hadoop YARN. The cluster has 6 nodes connected via 10 GigaBits per second Ethernet network. We use Pregel API [49] in Spark GraphX [48] framework to search the Action Graph in an iterative procedure by using the Algorithm described in Algorithm 12. This algorithm returns all low cost Action Rules ($cost < \phi$). From

Table 6.29: Parameters used for Action Rule discovery with SARGS and discovering low cost Action Rules

Property	**Car Evaluation Data**	**Mamm. Mass Data**	**Business Data**
Stable attributes	Doors, Persons	Age	Start Year, City, Sector, Specialization(SIC)
Required decision action	(Class) $unacc \rightarrow acc$	(Severity) $1 \rightarrow 0$	Estimated Sales $\$2M - \$10M \rightarrow \$10M - \$24M$
Minimum Support α and Confidence β	2, 70%	2, 70%	100, 70%
Cost Threshold ϕ	1500	2000	3000

these Action Rules, we do post processing step and highly correlating action terms pair ($correlation\ frequency \geq \eta$). If there is any correlation pair in the Action Rule, we drop the lowest cost in that pair.

A visualization of the Action Graph for the *Car Evaluation* data is shown in Figure 6.15. The color and size of the vertex v in the Action Graph represent how frequently a specific ActionTerm occurs in our Action Rules set. The more frequent ActionTerms are shown in larger size nodes, and the less frequent in smaller size nodes. Also, the darker colors signify the most frequently occuring ActionTerms, and the lighter colors less frequent ActionTerms. For the Car Evaluation data, the ActionTerm *(persons=more)* occurs most frequently, is shown in the red color *red node* in Figure 6.15 , and the ActionTerm $(safety, low \rightarrow high)$ is the second most frequent term in our Action Rules set, which is shown in green color the*green node*.

Table 6.30 gives details about the number of Action Rules, and the processing time in seconds for the proposed algorithm to build Action Graphs (one for each dataset) and basic properties of these graphs such as number of nodes and edges.

In Table 6.31, Table 6.32 and Table 6.33, we give sample actionable recommendations from our algorithms for finding low cost Action Rules for Car, Mammographic

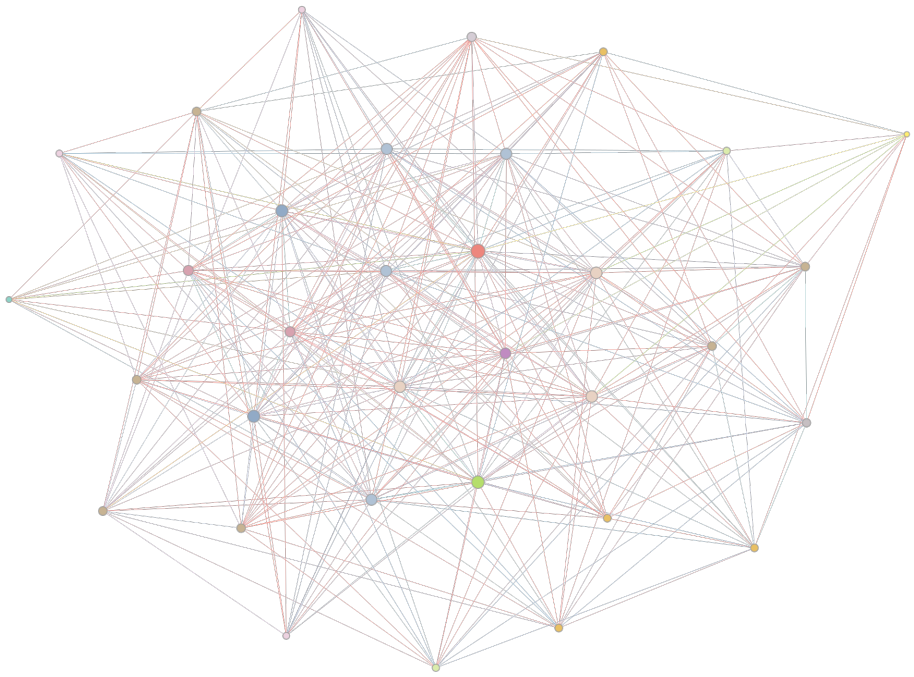


Figure 6.15: Action Graph for Action Rules from Car Evaluation Dataset

Mass and Business datasets respectively.

These Action Rules define what actions do a company/user should employ to achieve their desired goal. For example, the rule AR_{C1} recommends that if a car company increases the *Buying Cost* from *low* to *Very High* and increases the *Luggage Boot Size* from *Medium* to *Big* and lowers the *Maintenance Cost* from *Very High* to *Low* and increases *Safety Measures* from *Medium* to *High*, then the Car Condition may change from *Unacceptable* to *Acceptable* with the cost of 2200.0. For all datasets, we consider cost just as a measure of an Action Rule since the actual costs are assigned by experts.

In Table 6.34, we give our system's runtime performance comparing with non dis-

Table 6.30: Properties of Action Graphs

Property	**Car Evaluation Data**	**Mamm. Mass Data**	**Business Data**
No.of Action Rules	415	290	2043
No.of Action Terms / Nodes	33	98	224
No. of Edges	636	750 %	7390
Time taken to build Action Graph(secs)	7.2	7.3	9

tributed version of the same algorithm. The distributed version of the Action Graph, which we implement in Apache Spark [19] using the GraphX [48] library, and the Pregel API [49], shows faster processing times for large datasets compared to single machine implementation in Java.

6.12 Action Graph Search Algorithms - Experiment 2 - Action Graphs

To test the proposed methods, we use three datasets: *Car Evaluation* data, *Mammographic Mass* data, and the city of *Charlotte North Carolina BusinesWise* data, whose properties are given in Table 6.35

With our datasets, we run the SARGS algorithm on each data. We collect Action Rules which meet the minimum support(s), and minimum confidence(c), threshold. If we have n action terms in an Action Rule, we record the cost(ρ) for each n Action Terms. We calculate Total Cost of each Action Rule by adding the cost of all Action Terms in the rule. The cost of each action term is provided by a domain expert who has enough knowledge about the data. For example, for the Mammography dataset, a medical doctor specifies the cost for the suggested actions. For Car Evaluation data, the car manufacturer specifies the cost for the suggested actions. However, for our experiment purpose, we assign a random cost number to each ActionTerm. We assign the cost of *0* for Action Terms which have stable attributes, because the stable attributes cannot be changed. For the Flexible Attribute Action Terms the cost values are between 0 and 1. In Table 6.36, Table 6.37 and Table 6.38, we show

Table 6.31: Example Action Rules of Lowest Cost for Car Evaluation Dataset

1. $AR_{C1} : (buying, low \rightarrow vhigh) \wedge (lugBoot, med \rightarrow big) \wedge (maint, vhigh \rightarrow low) \wedge (persons = more) \wedge (safety, med \rightarrow high) \Longrightarrow (class, unacc \rightarrow acc)[Support : 4, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%]COST : 2200.0$

2. $AR_{C2} : (buying, low \rightarrow vhigh) \wedge (lugBoot, small \rightarrow big) \wedge (maint, med \rightarrow low) \wedge (persons = more) \wedge (safety, low \rightarrow high) \Longrightarrow (class, unacc \rightarrow acc)[Support : 4, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%]COST : 2100.0$

Low Cost Action Rules

1. $AR_{C4} : (buying, high \rightarrow med) \wedge (lugBoot, med \rightarrow small) \wedge (maint, low \rightarrow med) \wedge (persons = 4) \wedge (safety, low \rightarrow high) \Longrightarrow (class, unacc- > acc)[Support : 4, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%]COST : 1300.0$

2. $AR_{C5} : (buying, low \rightarrow med) \wedge (lugBoot, med \rightarrow big) \wedge (maint, low \rightarrow med) \wedge (persons = more) \wedge (safety, low \rightarrow med) \Longrightarrow (class, unacc- > acc)[Support : 4, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%]COST : 1400.0$

Low Cost Action Rules after Correlation

1. $AR_{C4} : (buying, high \rightarrow med) \wedge (lugBoot, med \rightarrow small) \wedge (maint, low \rightarrow med) \wedge (persons = 4) \wedge (safety, low \rightarrow high) \Longrightarrow (class, unacc- > acc)[Support : 4, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%]COST : 1000.0$

2. $AR_{C5} : (buying, low \rightarrow med) \wedge (lugBoot, med \rightarrow big) \wedge (maint, low \rightarrow med) \wedge (persons = more) \wedge (safety, low \rightarrow med) \Longrightarrow (class, unacc- > acc)[Support : 4, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%]COST : 1100.0$

Table 6.32: Example Action Rules of Lowest Cost for Mammographic Mass Data

High Cost Action Rules

1. $AR_{M1} : (BIRADS, 55 \rightarrow 4) \wedge (Density, 3 \rightarrow 2) \wedge (Shape, 4 \rightarrow 1) \Longrightarrow (Severity, 1 \rightarrow 0)[Support : 7, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%]COST : 1300.0$

2. $AR_{M2} : (BIRADS, 6 \rightarrow 4) \wedge (Margin, 5 \rightarrow 4) \wedge (Shape, 4 \rightarrow 1) \Longrightarrow (Severity, 1 \rightarrow 0)[Support : 3, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%]COST : 1200.0$

3. $AR_{M3} : (BIRADS, 5 \rightarrow 4) \wedge (Margin, 2 \rightarrow 4) \wedge (Shape, 4 \rightarrow 1) \Longrightarrow (Severity, 1 \rightarrow 0)[Support : 3, OldConfidence : 85\%, NewConfidence : 100\%, Utility : 85\%]COST : 1100.0$

Low Cost Action Rules

1. $AR_{M4} : (BIRADS, 6 \rightarrow 4) \wedge (Density, 3 \rightarrow 2) \wedge (Margin, 5 \rightarrow 1) \longrightarrow (Severity, 1 \rightarrow 0)[Support : 25, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%]COST : 550.0$

2. $AR_{M5} : (Age = 60) \wedge (BIRADS, 6 \rightarrow 4) \longrightarrow (Severity, 1 \rightarrow 0)[Support : 11, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%]COST : 450.0$

3. $AR_{M6} : (BIRADS, 6 \rightarrow 4) \wedge (Margin, 5 \rightarrow 1) \wedge (Shape, 3 \rightarrow 2) \longrightarrow (Severity, 1 \rightarrow 0)[Support : 13, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%]COST : 800.0$

Lowest Cost Action Rules after Correlation

1. $AR_{M4} : (BI - RADS, 6 \rightarrow 4) \wedge (Density, 3 \rightarrow 2) \wedge (Margin, 5 \rightarrow 1) \longrightarrow (Severity, 1 \rightarrow 0)[Support : 25, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%]COST : 450.0$

2. $AR_{M5} : (Age = 60) \wedge (BIRADS, 6 \rightarrow 4) \longrightarrow (Severity, 1 \rightarrow 0)[Support : 11, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%]COST : 400.0$

3. $AR_{M6} : (BIRADS, 6 \rightarrow 4) \wedge (Margin, 5 \rightarrow 1) \wedge (Shape, 3 \rightarrow 2) \longrightarrow (Severity, 1 \rightarrow 0)[Support : 13, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%]COST : 600.0$

Table 6.33: Example Action Rules of Lowest Cost for the Business Data

Low Cost Action Rules

1. AR_{B4} : $(EMPALLSITE, 5099Employees \rightarrow 250\text{-}499Employees) \wedge (EMPSITE, 5099Employees \rightarrow 4\text{-}9Employees) \wedge (SECTOR, Services \rightarrow RetailTrade) \wedge (STARTYR = 20062010) \Longrightarrow (ESTSALES, \$2M\text{-}\$10M \rightarrow \$10M\$24M)[Support : 2, OldConfidence : 60\%, NewConfidence : 66\%, Utility : 90\%]COST : 1615.0$

2. $AR_{B5} : (CITY, Matthews \rightarrow Charlotte) \wedge (EMPALLSITE, 25\text{-}49Employees \rightarrow 500999Employees) \wedge (EMPSITE, 4\text{-}9Employees \rightarrow 1024Employees) \wedge (OWNBLDG, Y\text{-} \rightarrow N) \Longrightarrow (ESTSALES, \$2M\$10M \rightarrow \$10M\$24M)[Support : 2, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%]COST : 1276.0$

Lowest Cost Action Rules after Correlation

1. AR_{B4} : $(EMPALLSITE, 5099Employees \rightarrow 250\text{-}499Employees) \wedge (EMPSITE, 5099Employees \rightarrow 4\text{-}9Employees) \wedge (SECTOR, Services \rightarrow RetailTrade) \wedge (STARTYR = 20062010) \Longrightarrow (ESTSALES, \$2M\text{-}\$10M \rightarrow \$10M\$24M)[Support : 2, OldConfidence : 60\%, NewConfidence : 66\%, Utility : 90\%]COST : 1330.0$

2. $AR_{B5} : (CITY, Matthews \rightarrow Charlotte) \wedge (EMPALLSITE, 25\text{-}49Employees \rightarrow 500999Employees) \wedge (EMPSITE, 4\text{-}9Employees \rightarrow 1024Employees) \wedge (OWNBLDG, Y\text{-} \rightarrow N) \Longrightarrow (ESTSALES, \$2M\$10M \rightarrow \$10M\$24M)[Support : 2, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%]COST : 1213.0$

Table 6.34: Analysis on Action Graphs

Dataset	**Non-distributed Algorithm**	**Distributed Algorithm**
Car Evaluation Data	1.2 mins	7.1 secs
Mamm. Mass Data	17 secs	5.8 secs
Business Data	>10 mins	3.1 mins

samples of high cost Action Rules, medium cost Action Rules and Low Cost Action Rules, for the *Car Evaluation Data*, *Mammographic Mass Data* and *Business Data* respectively. The high cost Action Rules are actions that the user would probably ignore since they are very high compared to the given cost threshold ρ. The user may or may not accept medium cost Action Rules because their costs are in the border of low cost and high cost Action Rules. The low cost Action Rules are more suitable for users since all low cost Action Rules are below the given cost threshold ρ. In Table 6.36, Table 6.37 and Table 6.38, low cost Action Rules are ones which has cost less than ρ.

These Action Rules define what actions do a company/user should employ to achieve their desired goal. For example, the rule AR_{C1} recommends that if a car company decreases the *Buying Cost* from *very high* to *low* and increases *luggage boot size* from *medium* to *big* and increases the *Maintenance Cost* from *low* to *very high* and increases *Safety Measures* from *low* to *medium* and if the *Seating Capacity* is *more than 4*, then the Car Condition may change from *Unacceptable* to *Acceptable* with the cost of 3.53. For all datasets, we consider cost just as a measure of an Action Rule since the actual costs are assigned by experts.

Next, we build an Action Graph using the list of extracted Action Rules as an input. We implement the Action Graph in both non-parallel environment, and in a clustered environment for performance and scalability comparision. The Non-parallel version is implemented in Java. The Apache Spark [19] using the Spark GraphX

library. We use Scala programming language. We test the system on a Spark cluster running over Hadoop YARN. The cluster has 6 nodes connected via 10 GigaBits per second Ethernet network. We then use algorithms 13 and 12 to return all low cost Action Rules ($cost < \rho$). From these Action Rules, we do post processing step and highly correlating action terms pair ($correlation\ frequency \geq \eta$). If there is any correlation pair in the Action Rule, we drop the lowest cost in that pair. For example, consider the low cost Action Rule AR_{B5} from $Table6$. Cost of this Action Rule is 1394.0. In the post-processing step, we find that the Action Term $(EMPSITE, 4-9Employees \rightarrow 10-24Employees)$ of cost 504.0 co-occurs frequently with the action term $(EMPALLSITE, 25-49Employees \rightarrow 500-999Employees)$ of cost 63.0. So we consider that one of these action terms trigger the other action to happen eventually. Thus, we drop the cost of $(EMPALLSITE, 25-49Employees \rightarrow 500-999Employees)$ and reduce the cost of the Action Rule to 1213.0.

A visualization of the Action Graph for the *Car Evaluation* data is shown in in Figure 6.15. The color and size of the vertex v in the Action Graph represent how frequently a specific ActionTerm occurs in our Action Rules set. The more frequent ActionTerms are shown in larger size nodes, and the less frequent in smaller size nodes. Also, the darker colors signify the most frequently occuring ActionTerms, and the lighter colors less frequent ActionTerms. For the Car Evaluation data, the Action-Term *(persons=more)* occurs most frequently, is shown in the red color *red node* in Figure 6.15, and the ActionTerm *(safety,low->high)* is the second most frequent term in our Action Rules set, which is shown in green color the*green node* in Figure 6.15.

Table 6.39 gives details about the number of Action Rules and basic properties of these graphs such as number of nodes, edges and number of connected component in Action Graphs and parameters set for algorithms to extract rules. It is notable that the Action Graph for the Business data is disconnected and contains 3 component in the graph. For experiment purpose, we set the minimum cost threshold ρ as 1.3 for

all datasets.

In Tables 6.40, 6.41 and 6.42, we give our system's runtime performance comparing with non distributed and distributed versions of the Dijkstra's shortest path, Breadth First Search and Depth First Search algorithms respectively for finding low cost Action Rules. The distributed version of the Action Graph, which we implement in Apache Spark [19] using the GraphX [48] library shows faster processing times for large datasets compared to single machine implementation in Java.

6.13 Action Graph Search Algorithms for Hospital Readmission

For all experiments, we use H-CUP data (described in Section 6.1.5) and we use vertical data distribution method to extract actionable patterns. We evaluate all action graph search methods: *Dijkstra's, Breadth First Search*, and *Depth First Search* algorithms to extract low cost action rules. Action rules from these graph search algorithms determine knowledge in the data that help hospitals to reduce readmissions at the lowest cost possible. For simplicity, we split the data by diagnosis - such that hospitals can actionable knowledge for the given disease. With this approach we add a concept of personalization towards diseases. In Table 6.43, we give a very short description about the data and also, we give parameters that we set for extracting action rules. Since actionable patterns require expert knowledge to assign cost, we assign random cost to these extracted patterns. For action term in the action rule, we assign the random cost values ranging from 0 to 1.

We show the Action Rules extracted using the vertical data distribution method for each diagnosed diseases in Table 6.44. Due to space constraints, we give action rules of only 2 disease codes: 227(Spinal cord injury) and 217(Anomalies during and before child birth). We consider these actionable patterns as recommendations to hospitals in such a way that for a given disease, if the hospital provides treatment or care for recommended diseases, hospitals can potentially reduce the number of hospital readmissions. For example, consider the action rule $H217_{AR2}$ that corresponds to disease

code *217*(Anomalies during and before child birth). The action rule recommends to the hospital that if hospitals prefer giving diagnosis for disease code *98*(Hypertension) in addition to treatment for disease code *217* and prefer treatment for disease code *138*(Ulcer) in addition to the code *58*(Nutritional disorders), hospitals can reduce readmission by 44%. The support of *35* shows that the framework identifies 81 entries in the data that supports condition. The total cost of this recommendation is *1.37*.

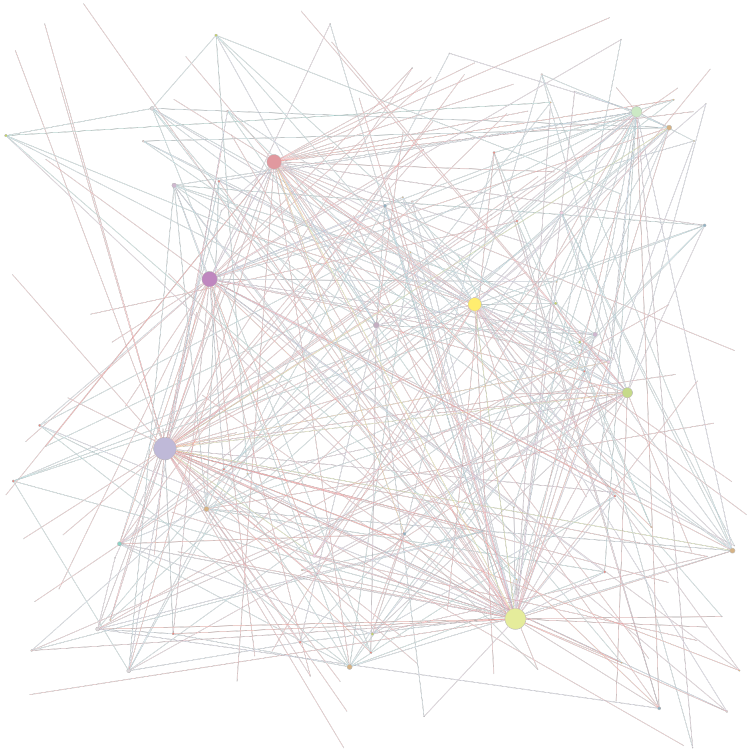


Figure 6.16: Sample Action Graph derived from Action Rules of Diagnosis 227 - Spinal Cord Injury

In Figures 6.16 and 6.17, we give sample action graphs built from actionable recommendations given by the knowledge extraction algorithm. Since the original action graph is much bigger for visualization, we used only a sample of the actionable patterns from the complete knowledge. The action graph of Diagnosis 217 comprise of *6626* nodes and *16288* edges and that Diagnosis 227 comprise of *831* nodes and

4671 edges. In these graphs, size and color of a node represents the frequency of the actionable pattern given by the recommender's system. Bigger the node, it has been used more in common with other actionable patterns and smaller nodes represent the actionable patterns have been recommended less frequently.

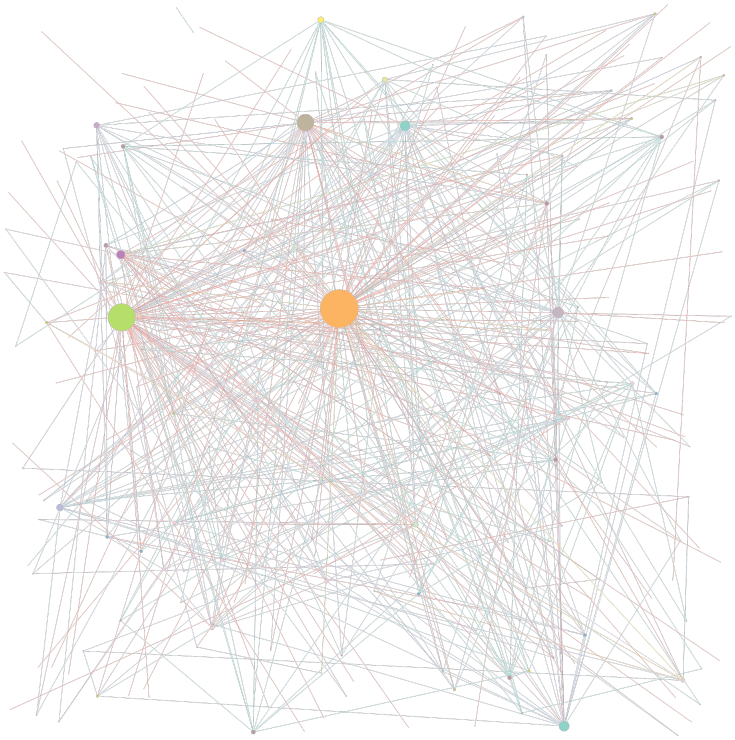


Figure 6.17: Sample action graph derived from action rules of Diagnosis 217 - Anomalies During and Before Child Birth

In the Table 6.45, we give action rules of lowest cost derived from action rules given in Table 6.44. It is notable from this recommendation that since the recommendations are made of 2 action terms, the recommender's system removes one of the terms to provide the low cost action rules.

In Tables 6.46 and 6.47, we give execution times of all the proposed action graph search algorithms for diagnosis codes 217 and 227 respectively. It is notable from these tables that Breadth First Search and Dijkstra's Shortes Path algorithms achieve much faster execution times(ateleast 2x times). On the other case Depth First Search

executes slower compared to the Java version of the algorithm. This is mainly because of complex parallelization approach followed by the algorithm to extract low cost action rules.

Table 6.35: Properties of the datasets

Property	**Car Evaluation Data**	**Mamm. Mass Data**	**Business Data**
# of instances	1728	961	22441
Attributes	7 attributes -Buying -Maintenance -Doors -Persons -Luggage Boot -Safety -Class	6 attributes -BI-RADS -Patient's age -Shape -Margin -Density -Severity	17 attributes including -City -Sector -Site Type -Building Type -Estimated Sales -Total Employees Count
Decision attribute values	Class (unacc, acc, good, vgood)	Severity (0 - benign, 1-malignant)	Estimated Sales (<$2M,2-10M,10-25M,25-50M,50-100M,100-500M,>500M)
# of instances / decision value	unacc - 1210 acc - 384 good - 69 vgood - 65	0 - 516 1 - 445	<$2M - 12503 $2-$10M - 1927 $10-$25M - 393 $25-$50M - 130 $50-$100M - 69 $100-$500M - 57 >$500M - 50
Data size	52 KB	16 KB	5.5 MB

Table 6.36: Example Action Rules of Lowest Cost, Medium Cost and High Cost for Car Evaluation Dataset

High Cost Action Rules

1. $AR_{C1} : (doors = 2) \wedge (lugBoot, big \rightarrow small) \wedge (maint, low \rightarrow vhigh) \wedge (persons, 2 \rightarrow more) \Longrightarrow (class, unacc \rightarrow acc)[Support : 4, OldConfidence : 25\%, NewConfidence : 100\%, Utility : 25\% COST : 3.523]$

2. $AR_{C2} : (buying, vhigh \rightarrow high) \wedge (lugBoot, med \rightarrow big) \wedge (maint, low \rightarrow med) \wedge (persons = 4) \wedge (safety, low \rightarrow med) \Longrightarrow (class, unacc \rightarrow acc)[Support : 4, OldConfidence : 25\%, NewConfidence : 100\%, Utility : 25\%, COST : 3.42]$

3. $AR_{C3} : (buying, high \rightarrow med) \wedge (lugBoot, small \rightarrow big) \wedge (maint, low \rightarrow high) \wedge (persons = more) \wedge (safety, med \rightarrow high) \Longrightarrow (class, unacc \rightarrow acc)[Support : 4, OldConfidence : 44\%, NewConfidence : 100\%, Utility : 44\%, COST : 3.049]$

Low Cost Action Rules

1. $AR_{C7} : (buying, high \rightarrow vhigh) \wedge (doors = 4) \wedge (lugBoot, big \rightarrow med) \wedge (maint, vhigh \rightarrow low) \wedge (persons = more) \wedge (safety, high \rightarrow med) \Longrightarrow (class, unacc \rightarrow acc)[Support : 1, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%, COST : 1.289]$

2. $AR_{C8} : (buying, vhigh \rightarrow low) \wedge (doors = 2) \wedge (lugBoot, big \rightarrow med) \wedge (maint, vhigh \rightarrow med) \wedge (persons = 4) \wedge (safety, high \rightarrow med) \Longrightarrow (class, unacc \rightarrow acc)[Support : 1, OldConfidence : 50\%, NewConfidence : 100\%, Utility : 50\%, COST : 1.095]$

3. $AR_{C9} : (buying, med \rightarrow high) \wedge (doors = 5more) \wedge (maint, med \rightarrow low) \wedge (persons = more) \wedge (safety, low \rightarrow high) \Longrightarrow (class, unacc \rightarrow acc)[Support : 3, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%, COST : 0.776]$

Table 6.37: Example Action Rules of Lowest Cost, Medium Cost and High Cost for Mammographic Mass Data

High Cost Action Rules

1. $AR_{M1} : (BI-RADS, 6 \rightarrow 4) \wedge (Margin, 5 \rightarrow 3) \wedge (Shape, 4 \rightarrow 2) \Longrightarrow (Severity, 1 \rightarrow 0)[Support : 13, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%, ; COST : 2.688]$

2. $AR_{M2} : (BI-RADS, 6 \rightarrow 4) \wedge (Density, 3 \rightarrow 2) \wedge (Shape, 4 \rightarrow 1) \Longrightarrow (Severity, 1 \rightarrow 0)[Support : 7, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%, COST : 2.683$

3. $AR_{M3} : (BI-RADS, 3 \rightarrow 4) \wedge (Margin, 5 \rightarrow 3) \wedge (Shape, 4 \rightarrow 2) \Longrightarrow (Severity, 1 \rightarrow 0)[Support : 13, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%, COST : 2.45$

Low Cost Action Rules

1. $AR_{M7} : (BI-RADS, 5 \rightarrow 2) \wedge (Density, 1 \rightarrow 3) \Longrightarrow (Severity, 1 \rightarrow 0)[Support : 6, OldConfidence : 80\%, NewConfidence : 100\%, Utility : 80\%, COST : 1.286$

2. $AR_{M8} : (BI-RADS, 6 \rightarrow 2) \wedge (Shape, 4 \rightarrow 2) \Longrightarrow (Severity, 1 \rightarrow 0)[Support : 2, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%, COST : 1.089$

3. $AR_{M9} : (Age = 63) \wedge (Margin, 5 \rightarrow 1) \Longrightarrow (Severity, 1 \rightarrow 0)[Support : 9, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%, COST : 0.783$

Table 6.38: Example Action Rules of Highest Cost and Lowest Cost for the Business Data

High Cost Action Rules

1. AR_{B1} : $(BLDGTYPE, Miscellaneous \rightarrow Office) \wedge (EMPALLSITE, 1-3Employees \rightarrow 10-24Employees) \wedge (EMPSITE, 1-3Employees \rightarrow 4-9Employees) \wedge (OWNBLDG, Y \rightarrow N) \wedge (SECTOR, Services \rightarrow WholesaleTrade) \wedge (SITETYPE, SingleSite \rightarrow Headquarters) \Longrightarrow (ESTSALES, \$2Mto\$10M \rightarrow \$10Mto\$25M)[Support : 2, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%, COST : 4.069$

2. AR_{B2} : $(CITY, Pineville \rightarrow Matthews) \wedge (INTLBUS, N \rightarrow Y) \wedge (OWNBLDG, N \rightarrow Y) \wedge (SECTOR, RetailTrade \rightarrow Manufacturing) \wedge (SITETYPE, Headquarters \rightarrow SingleSite) \Longrightarrow (ESTSALES, \$2Mto\$10M \rightarrow \$10Mto\$25M)[Support : 2, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%, COST : 3.969$

3. AR_{B3} : $(EMPALLSITE, 25-49Employees \rightarrow 50-99Employees) \wedge (EMPSITE, 4-9Employees \rightarrow 50-99Employees) \wedge (INTLBUS, Y \rightarrow N) \wedge (OWNBLDG, Y \rightarrow N) \wedge (SITETYPE, SingleSite \rightarrow Headquarters) \wedge (STARTYR = 1981-1990) \Longrightarrow (ESTSALES, \$2Mto\$10M \rightarrow \$10Mto\$25M)[Support : 2, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%, COST : 3.811$

Low Cost Action Rules

1. AR_{B7} : $(CITY, Matthews \rightarrow Cornelius) \wedge (EMPALLSITE, 50-99Employees \rightarrow 100-249Employees) \wedge (EMPSITE, 1-3Employees \rightarrow 50-99Employees) \Longrightarrow (ESTSALES, \$2Mto\$10M \rightarrow \$10Mto\$25M)[Support : 2, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%, COST : 1.071$

2. AR_{B8} : $(BLDGTYPE, Industrial \rightarrow Office) \wedge (CITY, Matthews \rightarrow Cornelius) \wedge (EMPSITE, 25-49Employees \rightarrow 50-99Employees) \Longrightarrow (ESTSALES, \$2Mto\$10M \rightarrow \$10Mto\$25M)[Support : 2, OldConfidence : 80\%, NewConfidence : 100\%, Utility : 80\%, COST : 0.827$

3. AR_{B9} : $(EMPALLSITE, 10-24Employees \rightarrow 100-249Employees) \wedge (SECTOR, I \rightarrow TransportationandPublicUtilities) \wedge (STARTYR = 1971-1980) \Longrightarrow (ESTSALES, \$2Mto\$10M \rightarrow \$10Mto\$25M)[Support : 2, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%, COST : 0.277$

Table 6.39: Analysis on Action Graphs

Property	**Car Evaluation Data**	**Mamm. Mass Data**	**Business Data**
Stable attributes	Doors, Buying	Age	Start Year
Required decision action	(Class) $unacc \rightarrow acc$	(Severity) $1 \rightarrow 0$	Estimated Sales $\$2M - \$10M \rightarrow \$10M - \$24M$
No.of Action Rules	82,503	552	191,934
No.of Action Terms / Nodes	45	112	188
Minimum Support s and Confidence c	1, 10%	2, 70%	2, 60%

Table 6.40: Runtimes of Dijkstra's shortest path algorithm on different datasets in seconds

Dataset	**Java Dijkstra's**	**Spark Dijkstra's**
Car Evaluation data	5s	2s
Mammographic Mass data	4s	4s
Business data	16s	10s

Table 6.41: Runtimes of Breadth First search algorithm on different datasets in seconds

Dataset	**Java BFS**	**Spark BFS**
Car Evaluation data	3s	2s
Mammographic Mass data	4s	4s
Business data	11s	7s

Table 6.42: Runtimes of Depth First search algorithm on different datasets in seconds

Dataset	**Java DFS**	**Spark DFS**
Car Evaluation data	4s	5s
Mammographic Mass data	5s	7s
Business data	18s	9s

Table 6.43: HCUP data attributes and Algorithm parameters

Property	**Description**
Attributes	67 attributes with DX(1-31) ; PR(1-31) ; Gender ; Race ; IsHomeless
Stable attributes	Gender ; Race ; IsHomeless ; PR(1-31)
Decision attribute	IsReadmitted
Required decision action	IsReadmitted($1 \rightarrow 0$)
Minimum support	30
Minimum confidence	40%
No. of diseases	262

Table 6.44: Sample Action Rules from the HCUP data for selected diagnosis

227(Spinal cord injury)

1. $H227_{AR1}$: $(DX, 237 \rightarrow 254) \wedge (DX, 199 \rightarrow 231) \Rightarrow (Readmission, 1- > 0)[Support : 40.0, OldConfidence : 56\%, NewConfidence : 64.58\%, 1.505]$

2. $H227_{AR2}$: $(DX, 237 \rightarrow 205) \wedge (DX, 244 \rightarrow 205) \Rightarrow (Readmission, 1 \rightarrow 0)[Support : 32.0, OldConfidence : 51.14\%, NewConfidence : 57.82\%, Cost : 1.465]$

3. $H227_{AR3}$: $(DX, 2 \rightarrow 254) \wedge (DX, 199 \rightarrow 155) \Rightarrow (Readmission, 1 \rightarrow 0)[Support : 39.0, OldConfidence : 43.01\%, NewConfidence : 51.62\%, Cost : 1.111]$

4. $H227_{AR4}$: $(DX, 199 \rightarrow 2) \wedge (DX, 159 \rightarrow 244) \Rightarrow (Readmission, 1 \rightarrow 0)[Support : 30.0, OldConfidence : 46.68\%, NewConfidence : 53.27\%, Cost : 1.253]$

217(Other congenital anomalies)

1. $H217_{AR1}$: $(DX, 217 \rightarrow 205) \wedge (DX, 217 \rightarrow 48) \Rightarrow (Readmission, 1 \rightarrow 0)[Support : 40.0, OldConfidence : 75.03\%, NewConfidence : 75.03\%, Cost : 1.627]$

2. $H217_{AR2}$: $(DX, 217 \rightarrow 98) \wedge (DX, 58 \rightarrow 138) \Rightarrow (Readmission, 1 \rightarrow 0)[Support : 35.0, OldConfidence : -44.28\%, NewConfidence : 51.46\%, Cost : 1.37]$

3. $H217_{AR3}$: $(DX, 213 \rightarrow 53) \wedge (DX, 213 \rightarrow 98) \Rightarrow (Readmission, 1 \rightarrow 0)[Support : 47.0, OldConfidence : 46.55\%, NewConfidence : 53.51\%, Cost : 1.085]$

4. $H217_{AR4}$: $(DX, 122 \rightarrow 205) \wedge (DX, 217 \rightarrow 58) \Rightarrow (Readmission, 1 \rightarrow 0)[Support : 39.0, OldConfidence : 55.99\%, NewConfidence : 61.68\%, Cost : 1.556]$

Table 6.45: Sample Low Cost Action Rules recommendations from the HCUP data for selected diagnosis

227(Spinal cord injury)

1. $H227_{AR1} : (DX, 237 \rightarrow 254) \Rightarrow (Readmission, 1- > 0)[Support : 40.0, OldConfidence : 56\%, NewConfidence : 64.58\%, 0.748]$
2. $H227_{AR2} : (DX, 237 \rightarrow 205) \Rightarrow (Readmission, 1 \rightarrow 0)[Support : 32.0, OldConfidence : 51.14\%, NewConfidence : 57.82\%, Cost : 0.984]$
3. $H227_{AR3} : (DX, 2 \rightarrow 254) \Rightarrow (Readmission, 1 \rightarrow 0)[Support : 39.0, OldConfidence : 43.01\%, NewConfidence : 51.62\%, Cost : 0.854]$
4. $H227_{AR4} : (DX, 159 \rightarrow 244) \Rightarrow (Readmission, 1 \rightarrow 0)[Support : 30.0, OldConfidence : 46.68\%, NewConfidence : 53.27\%, Cost : 0.651]$

217(Other congenital anomalies)

1. $H217_{AR1} : (DX, 217 \rightarrow 205) \Rightarrow (Readmission, 1 \rightarrow 0)[Support : 40.0, OldConfidence : 75.03\%, NewConfidence : 75.03\%, Cost : 0.634]$
2. $H217_{AR2} : (DX, 217 \rightarrow 98) \Rightarrow (Readmission, 1 \rightarrow 0)[Support : 35.0, OldConfidence : -44.28\%, NewConfidence : 51.46\%, Cost : 0.641]$
3. $H217_{AR3} : (DX, 213 \rightarrow 98) \Rightarrow (Readmission, 1 \rightarrow 0)[Support : 47.0, OldConfidence : 46.55\%, NewConfidence : 53.51\%, Cost : 0.63]$
4. $H217_{AR4} : (DX, 122 \rightarrow 205) \Rightarrow (Readmission, 1 \rightarrow 0)[Support : 39.0, OldConfidence : 55.99\%, NewConfidence : 61.68\%, Cost : 0.807]$

Table 6.46: Execution times of Action Graph search algorithms for Diagnosis 217

Algorithm	**Java execution**	**Spark execution**
Breadth First Search	8 mins	4.19 mins
Depth First Search	5.25 mins	4.2 mins
Dijkstra's Shortest Path	9.53 mins	3 mins

Table 6.47: Execution times of Action Graph search algorithms for Diagnosis 227

Algorithm	**Java execution**	**Spark execution**
Breadth First Search	4.62 mins	2.76 mins
Depth First Search	8 mins	6.5 mins
Dijkstra's Shortest Path	2 mins	1.15 mins

CHAPTER 7: CONCLUSIONS

7.1 MR-Random Forest

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

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

large amounts of data. Very limited work has been done on adapting Action Rules discovery to a distributed environment processing, therefore this study contributes to solving an important challenge.

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

7.2 SARGS

In this paper, we propose a new algorithm *Specific Action Rule discovery based on Grabbing Strategy (SARGS)* as an alternative to system ARoGS to extract complete Action Rules like system DEAR, ARED and Association Action Rules. We use this algorithm on a distributed cloud framework, Apache Spark, to efficiently produce Action Rules for the larger datasets. Our system outperforms the Hadoop MapReduce system for distributed Action Rules mining because of Spark's ability to perform in-memory computations and reduced communication cost compared to Hadoop MapReduce. We also suggest a more appropriate way to partition the data given to multiple nodes for Action Rules extraction.

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. Further we plan to test our system with more real-time large data like NPS dataset to improve system's scalability and feasibility. Finally, we plan to introduce the notion of cost of the Action Rules generated from the distributed environment.

7.3 Data Distribution

We propose two new methods for Data Distribution for Cloud Parallel processing , which can be applied to Actionable Pattern Mining via Action Rules. The Method 1 can be applied to most Action Rules algorithms, including ActionRules [27], system DEAR [27], ARAS [54].

Method 2 is specifically designed for Association Action rules [28], which is the most complex and time-consuming Action Rules extraction method, however it discovers all possible Action Rules. Previous works divide the data by random using default partitioning provided by Hadoop MapReduce, and Apache Spark. For that reason, the calculation of Support and Confidence may not represent very well the original support and confidence for ActionRules extracted on the entire dataset, or the support and confidence may be incorrect alltogether.

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

7.4 Vertical Data Distribution with Information Granules

Action Rules are recently being used in variety of domains like medicine, business and natural language processing. A distributed approach to derive Action Rules from the given data can benefit many such domains. In this work, we propose a novel method that divides the data using information granules rather than performing random distribution. This method provides more coherent optimization for data partitioning by taking correlations of granules with other granules. Combining this data

partitioning method with Action Rules extraction produces higher quality rules, with good processing time for larger datasets. The downside of the proposed approach can be taking combinations of rules from multiple partitions, which can become complex when each partition produce large number of rules. Thus the proposed method can be improved in the future by designing an approximation function for vertical data partitioning. The data partitioning can be done in such a way that we simply merge all rules from multiple partitions instead of combining or taking cartesian product of rules.

7.5 Semantic Data Distribution

In this section, we extend the algorithm from 4.4.2 by adding additional parameters to set privacy settings and load balancing to the actionable pattern extraction techniques. More importantly we propose a binary tree based load balancing module that split the data by attributes upto certain depth. Our results show that this method improves the algorithm performance in execution time for very large data like HCUP. Our analysis is the first effort to extract actionable recommendations for reducing hospital readmission in very efficient processing times and by utilizing data semantics.

Although our methods proved efficient in execution time, it is not very optimal in memory usage in the distributed setup. Also, our methods lack subject matter experts input to evaluate actionable recommendations. In future, we plan to address these problems by providing more optimal load balancing modules that are both memory and time optimal. We also plan to use experts input to evaluate our results.

7.6 Action Rules Cost in Spark

Considering the large volumes of patterns discovered by data mining methods, an interestingness measure is essential to filter out the patterns to the most useful ones. Action Rules mining discovers actionable patterns, which are considered interesting.

Little research has been done to measure interestingness of association rules or Action Rules. Action Rules of low cost are considered patterns of high interest, and as such are important.

The actions suggested by these special rules can be used for decision making purpose and to achieve the desired goals of the user or an organization. They can be applied in several domains including: medical, financial, industrial, educational, and social networks. However, with the advent of Big Data, the Action Rules algorithm require changes in order to adapt it to distributed environment processing and cloud computing. Current cloud computing frameworks offer a few adaptations for machine learning algorithms, however currently there do not exist any adaptations for Action Rules method or Action Rules lowest cost method.

In this work, we present an adaptation of Action Rules at lowest cost method to distributed processing using Apache Spark. The proposed adaptation improves the method, and acts as a scalable solution for producing low cost of Action Rules for large volumes of data at a reasonable processing time. The proposed approach outperforms the existing method in terms of computational efficiency with the Car Evaluation dataset and Mammographic dataset. In the future, we plan to improve the action set correlations matrix in order to reduce the cost, and perform additional experiments with financial data and social network data.

7.7 Action Graph

The distributed version of the Action Graph Search for Lowest Cost Action Rules, which we ipmlement in Apache Spark [19] using the GraphX [48] library, and the Pregel API [49], shows faster processing times for large datasets compared to single machine implementation in Java. Our proposed method presents an improvement over the Search for Action Rules of Lowest Costs in [4], as we use a distributed version for Graph Search, which is suitable to scale well for big datasets. In addition, it addresses a significant drawback of the previous method, which is using a heuristic

search, and hence sometimes it is unable to reach the goal, and discovery any rules. The new proposed method always reaches the goal, and discovers the rules of lowest cost. In the future, we plan to build a DecisionTree like structure, which can be searched, and shows the Lowest Cost Action Rules at the leaves of the tree. We plan to improve the support and confidence of the discovered Low Cost Acton Rules by incorporating these parameters into the search procedure. We also plan to use the proposed Graph structure in order to design a distributed version of Association Action Rules extraction algorithm.

7.8 Action Graph Search Algorithms

We study 3 methods for searching the Action Graph for Rules of Lowest Cost. We find that BFS and DFS perform better in terms of processing time, for large datasets when incorporated into parallel frameworks like Spark GraphX. For smaller datasets, all parallel algorithms perform almost similar to serialized versions of algorithms. However, the Dijkstra's Shortest Path discovers higher number of Low Cost Action Rules. We implement the DFS method in Spark GraphX, which has not been implemented before. For Action Graphs, we propose a modification of DFS algorithm to work similar to neighborhood aggregation. This is great advantage over previous implementations, as it allows for DFS search in very large graphs.

The proposed method presents a distributed version of the Action Graph Search for Lowest Cost Action Rules [4]. We ipmlement this method in Apache Spark [19] using the GraphX [48] library. This new method shows faster processing times for large datasets compared to single machine implementation in Java. Our proposed method presents an improvement over the Search for Action Rules of Lowest Cost in [4], as we use a distributed version for Graph Search, which is suitable to scale well for big datasets. In addition, it addresses a significant drawback of the previous method, which is using a heuristic search, and hence sometimes it is unable to reach the goal, and discovery any rules. The new proposed method always reaches the goal,

and discovers the rules of lowest cost.

7.9 Low Cost Action Rules for Hospital Readmission

In this section, we use the data provided by the HCUP [97] to extract actionable patterns for recommending insights to reduce hospital readmissions.

We incorporate the Action Graphs algorithm in 5.3 to action graphs and search for low cost actionable patterns from very large data. We use distributed computing tools like Apache Spark GraphX for fast and parallel graph search processing. We show that the proposed method can extract actionable patterns that potentially reduce the cost that the hospitals spend for reducing patient readmission. We also show that our method extracts these patterns within a span of seconds using the distributed processing tools.

Although our methods provide actionable recommendations to hospitals to reduce readmissions at lowest cost, the recommended actions require subject matter experts to analyze the recommendations, and provide their opinion. Since the recommendations involve the risk of lives of patients, it is best have the second opinion of medical expert to analyze these suggested actionable recommendations. This work can be extended in the future to engage experts or medical doctors input to evaluate and analyze the results.

CHAPTER 8: FUTURE WORK

In this work, we propose several algorithms to extract Actionable patterns from large datasets using distributed computing frameworks such as MapReduce[18] and Spark[19]. We also propose our initial steps on extracting low cost Action Rules and define the concept of Action Graph.

8.1 Granular Computing for Advanced Data Distribution

We propose an approach which uses Data Granules for Advanced data distribution for the vertical data partitioning methodology. This approach can ideally have much future work. We plan to consider the idea of studying the semantical relation between attributes, apart from just performing correlations between the attribute values. The application for this approach can be with the hospital readmission problem. Since in the hospital readmission problem the diagnoses and procedures are related to one another, considering such inter-relations would result in more intuitive recommendations for reducing hospital readmission. For example, instead of considering each diagnosis as a separate entity, if the algorithm considers the semantical relationaship or influence of one diagnosis over another, then the data can be split in more intelligent way. Hence, the problem becomes more fine grained and the results more accurate.

8.2 Vertical Data Distribution with Information Granules

This Information Granules approach has a shortcoming of taking combinations of rules from multiple partitions, which can increase the complexity when each partition produces large number of rules. We plan to improve this in the future by designing an approximation function for vertical data partitioning. The data partitioning can

be performed by merging all rules from multiple partitions instead of combining or taking the cartesian product of rules.

8.3 Semantic Data Distribution

The Semantic Data Distribution method is efficient in execution time, however it is not very optimal in memory usage in the distributed setup. In future, we plan to address this problem by providing more efficient load balancing modules that are both memory and time optimal. We also plan to use domain data experts input to evaluate our results.

8.4 Action Rules of Lowest Cost Graph

In the future we plan to improve our proposed Action Rules of Lowest Cost Graph method. The current method is using Dijkstra's shortest path method to find the Action Rule of lowest cost within the graph. We plan to propose a more advanced Heuristic Search Graph algorithm, and implement and test it using the Apache Spark GraphX library. We plan to evaluation the results with Large dataset.

8.5 Large Scale Experiments and Evaluation

In the future we plan to test our proposed methods with much larger datasets to evaluation computational tie and performance.

8.5.1 Amazon Product Recommendation

In the future we plan to apply our proposed Actionable Pattern mining methods to Amazon Product Recommendation data. We believe our proposed methods would extract interesting Actionable Patterns to suggest to users who search or purchase products on Amazon.com.

We plan to use the Amazon Bin Image Dataset [109], which is a public dataset provided by Amazon Web Services (AWS) [12]. It contains Over 500,000 bin JPEG images and corresponding JSON metadata files describing items in the bin. We plan to use the "aft-vbi-pds" S3 bucket, which is for the US East Region. The total overall

size of the data is 634.7 Gigabytes . We plan to utilize the AWS Cloud platform for accessing, and analyzing this large dataset in a distributed fashion.

8.5.2 Social Networks and Information Cascades

In the future we plan to experiment and apply the proposed Actionable Pattern mining methods to a Social Network data. We believe Actionable Pattern Recommendations can be applied to analyze Information Cascades in Social Networks. As well as our Large Graph Search algorithms, since the Social Networks can be represented in the form of a Large Graph. Since more people are actively participating in such social networks, they turn into a forum to innovate and exchange ideas and also help to influence our friends' behavior and make decisions [110]. Each social network node, which makes a decision, can affect its neighboring nodes for their decisions, and form a decision cascade [111]. Actionable patterns can be extracted to suggest actions to the node's advantage, such as: taking a flue vaccine, or recommending a hotel, or restaurant to friends.

All nodes in the network may have a fixed decision. However, this decision may change over time. Action Rules can provide actionable recommendations on what structural and temporal properties can trigger a change in a node's decision.

In the future, we plan to apply and evaluate our proposed methods for actionable pattern discovery on information cascades to the Live Journal Social Network dataset [112]. This is a free online network community with almost 10 million members, who are currently active and maintains journals and blogs. Out of them a significant number of people update their data for every 24 hours. This network comprises of 4.8 million nodes and 69 million edges.

Another dataset that can benefit from our proposed methods is Twitter network(s) [113]. This dataset defines information spreading patterns on Twitter during, before, and after an event. The event can be reading a news article. This dataset provides description of information propagation in terms of re-tweeting, replying and mentioning

patterns and also based on social relationships between users. The data was collected between 1st and 7th of July, 2012 and it contains 990,000 tweets from 456,000 users.

Since the Social Network data is very large in size, our proposed methods for distributed Actionable Pattern mining, and Large Graph Search, are essential for analyzing data of such size.

REFERENCES

[1] T. Silwattananusarn and K. Tuamsuk, "Data mining and its applications for knowledge management: a literature review from 2007 to 2012," *arXiv preprint arXiv:1210.2872*, 2012.

[2] Z. Cui, W. Chen, Y. He, and Y. Chen, "Optimal action extraction for random forests and boosted trees," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 179–188, ACM, 2015.

[3] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From data mining to knowledge discovery in databases," *AI magazine*, vol. 17, no. 3, p. 37, 1996.

[4] Z. W. Raś and A. A. Tzacheva, "In search for action rules of the lowest cost," in *Monitoring, Security, and Rescue Techniques in Multiagent Systems*, pp. 261–272, Springer, 2005.

[5] B. Padmanabhan and A. Tuzhilin, "On characterization and discovery of minimal unexpected patterns in rule discovery," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 2, pp. 202–216, 2006.

[6] R. L. Villars, C. W. Olofson, and M. Eastwood, "Big data: What it is and why you should care," *White Paper, IDC*, vol. 14, 2011.

[7] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. U. Khan, "The rise of "big data" on cloud computing: Review and open research issues," *Information Systems*, vol. 47, pp. 98–115, 2015.

[8] A. Bagavathi and A. Tzacheva, "Rule based systems in a 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)*, pp. 1–17, 2017.

[9] P. Mell, T. Grance, *et al.*, "The nist definition of cloud computing," 2011.

[10] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of internet services and applications*, vol. 1, no. 1, pp. 7–18, 2010.

[11] S. Pandey and S. Nepal, "Cloud computing and scientific applications—big data, scalable analytics, and beyond," *Future Generation Computer Systems*, vol. 7, no. 29, pp. 1774–1776, 2013.

[12] F. P. Miller, A. F. Vandome, and J. McBrewster, "Amazon web services," 2010.

[13] B. Wilder, "Cloud architecture patterns: using microsoft azure," 2012.

[14] I. S. Moreno, P. Garraghan, P. Townend, and J. Xu, "An approach for characterizing workloads in google cloud to derive realistic resource utilization models," 2013.

[15] A. Kochut, Y. Deng, M. R. Head, J. Munson, A. Sailer, H. Shaikh, C. Tang, A. Amies, M. Beaton, D. Geiss, *et al.*, "Evolution of the ibm cloud: Enabling an enterprise cloud services ecosystem," 2011.

[16] D. Talia, "Clouds for scalable big data analytics," *Computer*, vol. 46, no. 5, pp. 98–101, 2013.

[17] C. Ji, Y. Li, W. Qiu, U. Awada, and K. Li, "Big data processing in cloud computing environments," in *IEEE 12th International Symposium on Pervasive Systems, Algorithms and Networks (ISPAN)*, pp. 17–23, 2012.

[18] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[19] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and 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*, pp. 2–2, USENIX Association, 2012.

[20] J. Kuang, A. Daniel, J. Johnston, and Z. W. Raś, "Hierarchically structured recommender system for improving nps of a company," in *International Conference on Rough Sets and Current Trends in Computing*, pp. 347–357, Springer, 2014.

[21] M. Al-Mardini, A. Hajja, L. Clover, D. Olaleye, Y. Park, J. Paulson, and Y. Xiao, "Reduction of hospital readmissions through clustering based actionable knowledge mining," in *2016 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, pp. 444–448, 2016.

[22] A. Dardzinska, "Action rules mining in hoarseness disease," *Pattern and Data Analysis in Healthcare Settings*, p. 1, 2016.

[23] Z. W. Raś and A. Dardzińska, "From data to classification rules and actions," *International Journal of Intelligent Systems*, vol. 26, no. 6, pp. 572–590, 2011.

[24] Z. Pawlak and A. Skowron, "Rudiments of rough sets," *Information sciences*, vol. 177, no. 1, pp. 3–27, 2007.

[25] S. Kotsiantis and D. Kanellopoulos, "Association rules mining: A recent overview," *GESTS International Transactions on Computer Science and Engineering*, vol. 32, no. 1, pp. 71–82, 2006.

[26] S. Orlando, P. Palmerini, and R. Perego, "Enhancing the apriori algorithm for frequent set counting," in *DaWaK*, vol. 1, pp. 71–82, Springer, 2001.

[27] Z. W. Ras and A. Wieczorkowska, "Action-rules: How to increase profit of a company," in *European Conference on Principles of Data Mining and Knowledge Discovery*, pp. 587–592, Springer, 2000.

[28] Z. W. Ras, A. Dardzinska, L.-S. Tsay, and H. Wasyluk, "Association action rules," in *IEEE International Conference on Data Mining Workshops, 2008. ICDMW'08.*, pp. 283–290, 2008.

[29] A. A. Tzacheva, C. C. Sankar, S. Ramachandran, and R. A. Shankar, "Support confidence and utility of action rules triggered by meta-actions," in *IEEE International Conference on Knowledge Engineering and Applications (ICKEA)*, pp. 113–120, 2016.

[30] M.-Y. Chang, R.-D. Chiang, S.-J. Wu, and C.-H. Chan, "Mining unexpected patterns using decision trees and interestingness measures: a case study of endometriosis," *Soft Computing*, vol. 20, no. 10, pp. 3991–4003, 2016.

[31] A. Dardzinska, *Action rules mining*, vol. 468. Springer, 2012.

[32] A. Dardzińska and Z. W. Raś, "Cooperative discovery of interesting action rules," in *International Conference on Flexible Query Answering Systems*, pp. 489–497, Springer, 2006.

[33] A. A. Tzacheva and L.-S. Tsay, "Tree-based construction of low-cost action rules," *Fundamenta Informaticae*, vol. 86, no. 1, 2, pp. 213–225, 2008.

[34] H. Touati, Z. W. Raś, J. Studnicki, and A. A. Wieczorkowska, "Mining surgical meta-actions effects with variable diagnoses' number," in *International Symposium on Methodologies for Intelligent Systems*, pp. 254–263, Springer, 2014.

[35] A. A. Tzacheva and Z. W. Ras, "Association action rules and action paths triggered by meta-actions," in *2010 IEEE International Conference on Granular Computing (GrC)*, pp. 772–776, 2010.

[36] K. Wang, Y. Jiang, and A. Tuzhilin, "Mining actionable patterns by role models," in *IEEE Proceedings of the 22nd International Conference on Data Engineering, 2006. ICDE'06.*, pp. 16–16, 2006.

[37] J. Li, Y. Ren, C. Mei, Y. Qian, and X. Yang, "A comparative study of multigranulation rough sets and concept lattices via rule acquisition," *Knowledge-based systems*, vol. 91, pp. 152–164, 2016.

[38] Y. Chen, Q. Zhu, and H. Xu, "Finding rough set reducts with fish swarm algorithm," *Knowledge-Based Systems*, vol. 81, pp. 22–29, 2015.

[39] J. T. Yao, A. V. Vasilakos, and W. Pedrycz, "Granular computing: perspectives and challenges," *IEEE Transactions on Cybernetics*, vol. 43, no. 6, pp. 1977–1989, 2013.

[40] L. A. Zadeh, "Toward a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic," *Fuzzy sets and systems*, vol. 90, no. 2, pp. 111–127, 1997.

[41] X. Wu, W. Fan, J. Peng, K. Zhang, and Y. Yu, "Iterative sampling based frequent itemset mining for big data," *International Journal of Machine Learning and Cybernetics*, vol. 6, no. 6, pp. 875–882, 2015.

[42] Y. Yao, "A triarchic theory of granular computing," *Granular Computing*, vol. 1, no. 2, pp. 145–157, 2016.

[43] S. García, J. Luengo, and F. Herrera, *Data preprocessing in data mining*. Springer, 2016.

[44] R. Bro and A. K. Smilde, "Principal component analysis," *Analytical Methods*, vol. 6, no. 9, pp. 2812–2831, 2014.

[45] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *2010 IEEE 26th symposium on Mass storage systems and technologies (MSST)*, pp. 1–10, 2010.

[46] C. Lam, *Hadoop in action*. Manning Publications Co., 2010.

[47] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, *et al.*, "Mllib: Machine learning in apache spark," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[48] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "Graphx: Graph processing in a distributed dataflow framework.," in *OSDI*, vol. 14, pp. 599–613, 2014.

[49] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pp. 135–146, ACM, 2010.

[50] Y. Low, J. E. Gonzalez, A. Kyrola, D. Bickson, C. E. Guestrin, and J. Hellerstein, "Graphlab: A new framework for parallel machine learning," *arXiv preprint arXiv:1408.2041*, 2014.

[51] C. Avery, "Giraph: Large-scale graph processing infrastructure on hadoop," *Proceedings of the Hadoop Summit. Santa Clara*, vol. 11, no. 3, pp. 5–9, 2011.

[52] M. Xu, Y. Liu, Q. Huang, Y. Zhang, and G. Luan, "An improved dijkstra's shortest path algorithm for sparse network," *Applied Mathematics and Computation*, vol. 185, no. 1, pp. 247–254, 2007.

[53] L.-S. Tsay and Z. W. Raś, "Action rules discovery system dear_3," in *International Symposium on Methodologies for Intelligent Systems*, pp. 483–492, Springer, 2006.

[54] Z. W. Raś, E. Wyrzykowska, and H. Wasyluk, "Aras: Action rules discovery based on agglomerative strategy," in *International Workshop on Mining Complex Data*, pp. 196–208, Springer, 2007.

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

[56] A. Hajja and Z. W. Ras, "Object-driven action rules," in *Encyclopedia of Information Science and Technology, Third Edition*, pp. 1197–1206, IGI Global, 2015.

[57] J. W. Grzymała-Busse, S. R. Marepally, and Y. Yao, "An empirical comparison of rule sets induced by lers and probabilistic rough classification," *Rough Sets and Intelligent Systems-Professor Zdzisław Pawlak in Memoriam*, pp. 261–276, 2013.

[58] J. Rauch and M. Simunek, "Action rules and the guha method: Preliminary considerations and results," *Foundations of Intelligent Systems*, pp. 76–87, 2009.

[59] A. A. Tzacheva, R. A. Shankar, S. Ramachandran, and A. Bagavathi, "Action rules of lowest cost and action set correlations," *Fundamenta Informaticae Journal,European Association for Theoretical Computer Science (EATCS), IOS Press*, 2017.

[60] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," 1991.

[61] G. Wu, H. Li, X. Hu, Y. Bi, J. Zhang, and X. Wu, "Mrec4. 5: C4. 5 ensemble classification with mapreduce," in *ChinaGrid Annual Conference, 2009. ChinaGrid'09. Fourth*, pp. 249–255, IEEE, 2009.

[62] W. Dai and W. Ji, "A mapreduce implementation of c4. 5 decision tree algorithm," *International journal of database theory and application*, vol. 7, no. 1, pp. 49–60, 2014.

[63] V. Kolias, C. Kolias, I. Anagnostopoulos, and E. Kayafas, "Rulemr: Classification rule discovery with mapreduce," in *IEEE International Conference on Big Data (Big Data)*, pp. 20–28, 2014.

[64] X. Lin, "Mr-apriori: Association rules algorithm based on mapreduce," in *5th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pp. 141–144, 2014.

[65] H. Qiu, R. Gu, C. Yuan, and Y. Huang, "Yafim: a parallel frequent itemset mining algorithm with spark," in *2014 IEEE International Parallel & Distributed Processing Symposium Workshops (IPDPSW)*, pp. 1664–1671, 2014.

[66] S. Rathee, M. Kaul, and 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*, pp. 27–34, ACM, 2015.

[67] P.-N. Tan, M. Steinbach, and V. Kumar, "Classification: basic concepts, decision trees, and model evaluation," *Introduction to data mining*, vol. 1, pp. 145–205, 2006.

[68] V. Nikam and B. Meshram, "Parallel and scalable rules based classifier using map-reduce paradigm on hadoop cloud," *International Journal of Advanced Technology in Engineering and Science*, vol. 2, no. 08, pp. 558–568, 2014.

[69] K. Adhatrao, A. Gaykar, A. Dhawan, R. Jha, and V. Honrao, "Predicting students' performance using id3 and c4. 5 classification algorithms," *arXiv preprint arXiv:1310.2071*, 2013.

[70] H. Liu and A. Gegov, "Collaborative decision making by ensemble rule based classification systems," in *Granular Computing and Decision-Making*, pp. 245–264, Springer, 2015.

[71] A. Skowron and J. Stepaniuk, "Modeling of high quality granules," in *International Conference on Rough Sets and Intelligent Systems Paradigms*, pp. 300–309, Springer, 2007.

[72] V. Kreinovich, "Interval computations as an important part of granular computing: an introduction," *Handbook of Granular Computing*, pp. 1–31, 2008.

[73] H. Li, L. Zhang, B. Huang, and X. Zhou, "Sequential three-way decision and granulation for cost-sensitive face recognition," *Knowledge-Based Systems*, vol. 91, pp. 241–251, 2016.

[74] S. Liu, W. Pedrycz, A. Gacek, and Y. Dai, "Development of information granules of higher type and their applications to granular models of time series," *Engineering Applications of Artificial Intelligence*, vol. 71, pp. 60–72, 2018.

[75] A. A. Tzacheva, A. Bagavathi, and P. D. Ganesan, "Mr–random forest algorithm for distributed action rules discovery," *International Journal of Data Mining and Knowledge Management Process*, pp. 15–30, 2016.

[76] A. Bagavathi, P. Mummoju, K. Tarnowska, A. A. Tzacheva, and Z. W. Ras, "Sargs method for distributed actionable pattern mining using spark," in *2017 IEEE International Conference on Big Data (Big Data)*, pp. 4272–4281, Dec 2017.

[77] E. Liberty, K. Lang, and K. Shmakov, "Stratified sampling meets machine learning," in *International Conference on Machine Learning*, pp. 2320–2329, 2016.

[78] A. Bagavathi, V. Rao, and A. A. Tzacheva, "Data distribution method for scalable actionable pattern mining," in *Proceedings of the First International Conference on Data Science, E-learning and Information Systems*, DATA '18, (New York, NY, USA), pp. 3:1–3:7, ACM, 2018.

[79] M. Hahsler and R. Karpienko, "Visualizing association rules in hierarchical groups," *Journal of Business Economics*, vol. 87, no. 3, pp. 317–335, 2017.

[80] A. Bagavathi, A. Tripathi, A. A. Tzacheva, and Z. W. Ras, "Actionable pattern mining - a scalable data distribution method based on information granules," in *IEEE 17th International Conference on Machine Learning and Applications (ICMLA'18)*, pp. 32–39, Dec 2018.

[81] E. J. Ghomi, A. M. Rahmani, and N. N. Qader, "Load-balancing algorithms in cloud computing: a survey," *Journal of Network and Computer Applications*, vol. 88, pp. 50–71, 2017.

[82] K. Nishant, P. Sharma, V. Krishna, C. Gupta, K. P. Singh, R. Rastogi, *et al.*, "Load balancing of nodes in cloud using ant colony optimization," in *IEEE 14th International Conference on Modelling and Simulation*, pp. 3–8, 2012.

[83] P. V. Krishna, "Honey bee behavior inspired load balancing of tasks in cloud computing environments," *Applied Soft Computing*, vol. 13, no. 5, pp. 2292–2303, 2013.

[84] Z. W. Ras, A. A. Tzacheva, L.-S. Tsay, and O. Giirdal, "Mining for interesting action rules," in *IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pp. 187–193, 2005.

[85] Q. Yang, J. Yin, C. Ling, and R. Pan, "Extracting actionable knowledge from decision trees," *IEEE Transactions on Knowledge and data Engineering*, vol. 19, no. 1, pp. 43–56, 2007.

[86] M. Karim and R. M. Rahman, "Decision tree and naive bayes algorithm for classification and generation of actionable knowledge for direct marketing," *Journal of Software Engineering and Applications*, vol. 6, no. 04, p. 196, 2013.

[87] P. Su, D. Li, and K. Su, "An expected utility-based approach for mining action rules," in *Proceedings of the ACM SIGKDD Workshop on Intelligence and Security Informatics*, ISI-KDD '12, (New York, NY, USA), pp. 9:1–9:4, ACM, 2012.

[88] Y. Hu, J. Du, X. Zhang, X. Hao, E. Ngai, M. Fan, and M. Liu, "An integrative framework for intelligent software project risk planning," *Decision Support Systems*, vol. 55, no. 4, pp. 927–937, 2013.

[89] Y. Hu, B. Feng, X. Mo, X. Zhang, E. Ngai, M. Fan, and M. Liu, "Cost-sensitive and ensemble-based prediction model for outsourced software project risk prediction," *Decision Support Systems*, vol. 72, pp. 11–23, 2015.

[90] A. A. Tzacheva, A. Bagavathi, and S. C. Suryanarayanaprasad, "In search of actionable patterns of lowest cost–a scalable action graph method," in *2018 IEEE First International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pp. 119–124, 2018.

[91] M. M. Rathore, A. Ahmad, A. Paul, and G. Jeon, "Efficient graph-oriented smart transportation using internet of things generated big data," in *IEEE 2015 11th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*, pp. 512–519, 2015.

[92] A. A. Tzacheva, A. Bagavathi, and S. C. Suryanarayanaprasad, "In search of actionable patterns of lowest cost–a scalable action graph method," in *IEEE First International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pp. 119–124, 2018.

[93] G. Chu, C. Schulte, and P. J. Stuckey, "Confidence-based work stealing in parallel constraint programming," in *International conference on principles and practice of constraint programming*, pp. 226–241, Springer, 2009.

[94] M. Naumov, A. Vrielink, and M. Garland, "Parallel depth-first search for directed acyclic graphs," in *Proceedings of the Seventh Workshop on Irregular Applications: Architectures and Algorithms*, p. 4, ACM, 2017.

[95] M. Lichman, "Uci machine learning repository," tech. rep., Irvine, CA, USA, 2013.

[96] Z. W. Ras, K. A. Tarnowska, J. Kuang, L. Daniel, and D. Fowler, "User friendly nps-based recommender system for driving business revenue," in *International Joint Conference on Rough Sets*, pp. 34–48, Springer, 2017.

[97] A. for Healthcare Research and M. Quality, Rockville, "Hcup state inpatient databases (nrd). healthcare cost and utilization project (hcup). 2011-2012," 2011-2012.

[98] S. F. Jencks, M. V. Williams, and E. A. Coleman, "Rehospitalizations among patients in the medicare fee-for-service program," *New England Journal of Medicine*, vol. 360, no. 14, pp. 1418–1428, 2009.

[99] R. B. Zuckerman, S. H. Sheingold, E. J. Orav, J. Ruhter, and A. M. Epstein, "Readmissions, observation, and the hospital readmissions reduction program," *New England Journal of Medicine*, vol. 374, no. 16, pp. 1543–1551, 2016.

[100] N. Esfandiari, M. R. Babavalian, A.-M. E. Moghadam, and V. K. Tabar, "Knowledge discovery in medicine: Current issue and future trend," *Expert Systems with Applications*, vol. 41, no. 9, pp. 4434–4463, 2014.

[101] R. S. Baker and P. S. Inventado, "Educational data mining and learning analytics," in *Learning analytics*, pp. 61–75, Springer, 2014.

[102] G. Shmueli, P. C. Bruce, I. Yahav, N. R. Patel, and K. C. Lichtendahl Jr, *Data mining for business analytics: concepts, techniques, and applications in R*. John Wiley & Sons, 2017.

[103] H. C. Koh, G. Tan, *et al.*, "Data mining applications in healthcare," *Journal of healthcare information management*, vol. 19, no. 2, p. 65, 2011.

[104] D. He, S. C. Mathews, A. N. Kalloo, and S. Hutfless, "Mining high-dimensional administrative claims data to predict early hospital readmissions," *Journal of the American Medical Informatics Association*, vol. 21, no. 2, pp. 272–279, 2014.

[105] B. Zheng, J. Zhang, S. W. Yoon, S. S. Lam, M. Khasawneh, and S. Poranki, "Predictive modeling of hospital readmissions using metaheuristics and data mining," *Expert Systems with Applications*, vol. 42, no. 20, pp. 7110–7120, 2015.

[106] B. Strack, J. P. DeShazo, C. Gennings, J. L. Olmo, S. Ventura, K. J. Cios, and J. N. Clore, "Impact of hba1c measurement on hospital readmission rates: analysis of 70,000 clinical database patient records," *BioMed research international*, vol. 2014, 2014.

[107] P. Braga, F. Portela, M. F. Santos, and F. Rua, "Data mining models to predict patient's readmission in intensive care units," in *ICAART 2014-Proceedings of the 6th International Conference on Agents and Artificial Intelligence*, vol. 1, pp. 604–610, SCITEPRESS, 2014.

[108] M. Almardini, A. Hajja, Z. W. Raś, L. Clover, D. Olaleye, Y. Park, J. Paulson, and Y. Xiao, "Reduction of readmissions to hospitals based on actionable knowledge discovery and personalization," in *Beyond Databases, Architectures and Structures. Advanced Technologies for Data Mining and Knowledge Discovery*, pp. 39–55, Springer, 2015.

[109] A. Zeng, K.-T. Yu, S. Song, D. Suo, E. Walker, A. Rodriguez, and J. Xiao, "Multi-view self-supervised deep learning for 6d pose estimation in the amazon picking challenge," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1386–1383, 2017.

[110] R. Alhajj and J. Rokne, *Encyclopedia of social network analysis and mining*. Springer Publishing Company, Incorporated, 2014.

[111] W. Shu and Y.-H. Chuang, "The perceived benefits of six-degree-separation social networks," *Internet Research*, vol. 21, no. 1, pp. 26–45, 2011.

[112] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters," *Internet Mathematics*, vol. 6, no. 1, pp. 29–123, 2009.

[113] M. De Domenico, A. Lima, P. Mougel, and M. Musolesi, "The anatomy of a scientific rumor," *Scientific reports*, vol. 3, p. 2980, 2013.