12-2021

Rare Action Rule Exploration

Blake A. Johns
University of South Alabama, baj1621@jagmail.southalabama.edu

RARE ACTION RULE EXPLORATION

A Thesis

Submitted to the Graduate Faculty of the
University of South Alabama
in partial fulfillment of the
requirements for the degree of

Master of Science

in

Computer and Information Sciences

by
Blake A. Johns
B. S., University of South Alabama, 2020
December 2021

ACKNOWLEDGMENTS

I would like to thank my wife Kelcie for standing by my side throughout my undergraduate and graduate career. Without her support I would have never been able to make it as far as I have in academics and in life. I would also like to thank the employees of the coffee shop in Shelby Hall. The moral support in both words and caffeine kept my spirits high throughout my adventure. I would like to thank Dr. Johnsten, Dr. Bourrie, and Dr. Colarusso for the advice they gave and their willingness to serve as a part of my thesis committee. Finally, I would like to thank Dr. Benton for taking a student who admitted he knew nothing and giving him a chance to learn and for continuing to push me to new heights that I could not yet see.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AAA……Attribute Analysis Algorithm

AAR……..Association Action Rules

ARED……Action Rule Extraction from Decision Table

ARD……..Action Rule Discovery

DEAR……...Discovering Extended Action Rules

FAARM……Frequent Association Action Rules Mining

FAARM_nS…….Frequent Association Action Rules Mining with No Inner Support

FAARM_S……Frequent Association Action Rules Mining with All Support

FR……Frequent to Rare

KDD……..Knowledge Discovery in Databases

MARFS…….Mining Action Rules From Scratch

MOEAR……...Multi-Objective Evolutionary Action Rules

RARE…….Rare Action Rule Exploration

RF……Rare to Frequent

RR……..Rare to Rare

ABSTRACT

Johns, Blake, A., M.S., University of South Alabama, December 2021. Rare Action Rule Exploration. Chair of Committee: Ryan Benton, Ph.D.

Action rules describe a system's required transitions to achieve a desired class transition. Composed of stable attributes (such as age) and flexible attributes (such as interest rate), these transition observations are particularly interesting because they can give insight on a change in the system such as going from an unfavorable state to a favorable one. In the domain of action rules, what is considered rare has not been formally defined. To form a definition for rare action rules, we investigate the way that rarity has been defined in a similar domain: association rule mining. Association rule mining is a pattern mining technique that generates association rules which describe object relationships, or correlations, with one another. Although rarity has numerous definitions in this realm, they do not translate directly into action rules. This research proposes a definition for rarity in action rules and two algorithms: a consequent constraint algorithm to generate rare action rules and an attribute analysis algorithm. The rule generation algorithm utilizes a user provided consequent paired with support to generate the rare action rules and the attribute analysis algorithm uses confidence to identify potentially interesting attribute transitions with respect to a class transition.

CHAPTER I

INTRODUCTION

With advancements in technology, information is becoming more digitized. Uncovering the knowledge in this information is known as the Knowledge Discovery in Databases (KDD) process and involves five major steps: data selection, data preprocessing, transformation, data mining, and evaluation [1]. Data mining involves the application of machine learning algorithms onto the data in hopes to reveal interesting patterns [1]. These algorithms encapsulate four major techniques: regression, association rule discovery, classification, and clustering. The two techniques that are relevant to action rules are association rule discovery and classification. Association rule discovery focuses on identifying correlations between objects within the data [2]. These correlations have two general forms; frequent patterns, which are correlations that occur within a large percentage of the data and infrequent patterns which are the statistical complement [3]. Classification techniques attempt to find a model or function that fits a class or type of object(s) or data [3]. These methods focus on prediction with a certain degree of accuracy.

Action rules are constructed of stable (unchangeable), flexible (changeable), and class attributes and were introduced by Ras and Dardzinska [4]. These rules attempt to assist in the decision-making process by supplying information on the transitions of the attributes in the system as correlated with the transition of the class. This information

would be in the form of an attribute transition that influenced the class attribute's transition [4]. These action rules are typically built from the extraction of classification rules [4], association rules [5], or directly from the data [6].

Most of the work done in action rules focus on frequently occurring action rules, or rules that meet a given support. The rules that do not adhere to this can be considered rare and have not seen many research efforts nor formalization. This research focuses on rare action rules to develop a specific definition of what a rare action rule is, an algorithm to generate these rules, and an attribute analysis algorithm to identify potentially interesting attribute transitions within the generated rules. Before the method of how to find them is discussed, rare action rules must be defined. Currently, there has not been much work on these definitions other than acknowledging rarity exist, where something is considered rare if they do not meet the requirements to be frequent (e.g., an occurrence of more than 60 percent is considered frequent and anything under this percentage is considered rare) [7]. Because of this, the definitions of rarity utilized by association rule discovery techniques are identified, analyzed, and mapped into the domain of action rules. An algorithm for generating the rare action rules will be developed based on this definition. The proposed algorithm adapts an association rule-based approach that utilizes a consequent constraint search technique. This ensures the results of the research will generate only rare rules containing a specified outcome.

This thesis is organized as follows. Chapter II will provide a detailed review of the literature on both action rules and rare association rules, Chapter III will discuss the steps of the methodology, Chapter IV will disclose the experiments, Chapter V will

provide the results of the experiments and discuss the conclusions and the future work for the research.

CHAPTER II

LITERATURE REVIEW

The work reported on in the literature for action rule mining utilizes classification and association rule discovery techniques to generate action rules. Action rules were proposed as an attempt to increase the profits of a company through explicit changes to attributes to move a class state from unfavorable profits to favorable profits by Ras and Wieczorkowska [4]. Action rules focus on what attribute changes would influence a change with a class to move from an undesirable state to a desirable state [4]. He, Xu, Deng, and Ma [6] suggest the usage of association-inspired approaches with the introduction of support in action rules to get away from classification-based generation. Support is the frequency of an item's appearance in the data compared to the number of instances in the data. The introduction of the support metric brings the concept of a frequent action rule into play where an action rule is frequent if it exceeds the minimum support threshold [6]. The support measure is used as a pruning threshold continues to be adopted in the search of frequent action rules; as a result, action rules that do not meet this criterion are abandoned due to the lack of value to the user [5-11]. However, the discarded action rules may have the potential to hold critical information.

To build a definition for a rare action rule, literature in rare association rule discovery is reviewed. Association rules were first presented by Hájek, Havel, and Chytil

[12] but was widely popularized by Agrawal, Imielinski, and Swami [2]. An association rule is described as object correlations such as the purchase of one or more known objects implies the purchase of another object. Association rule mining utilizes a support metric to generate frequent rules in the data [2]. Although initially the rules that did not meet these requirements were handled in the same manner as in action rule mining, there has been significant work handling rarity in association rule discovery.

This literature review will follow the following organization. Section 2.2 establishes the key definitions used in both action rules and association rules. Section 2.3 discusses the work done in generating action rules using extracted classification rules. Section 2.4 discusses the work done in generating action rules directly from the data. Finally, Section 2.5 discusses the work done in generating rare association rules.

2.1 An Overview of Definitions

An action rule is composed of 3 components: stable attributes, flexible attributes, and a class attribute [4]. A stable attribute is defined as an attribute that cannot undergo a change [4], such as a date of birth. A flexible attribute is an attribute that can change [4], such as the interest rate on a loan for a customer. A class attribute is an attribute that is seen as a state (decision) [4], such as loan approval. The formal definition of action rules is noted as $\left[(b_1, v_1 \rightarrow w_1) * (b_2, v_2 \rightarrow w_2) * \ldots * \left(b_p, v_p \rightarrow w_p\right)\right] \Rightarrow [(d, k_1) \rightarrow (d, k_2)]$, where *b* is the attribute, *v* is the attribute's initial state, *w* is the attribute's changed state (from $v \rightarrow w$), *d* is the class attribute, and *k* is the class attribute's transition [4]. An example of this definition would be $\left[\left(client_{age}, 50\right) * (client_{taxBracket}, 22\% \rightarrow\right.$

$24\%)\big] \Rightarrow (bank_{loan}, small \rightarrow large)$ where the client's age is a stable attribute, the client tax bracket increased from 22% to 24% which implies the bank can adjust the awarded loan amount from small to large.

The introduction of action rules as an association rule discovery problem brought a support measurement into use [5]. Support is notably seen in association rule discovery and is defined as $\frac{count(x)}{D}$, where *x* is the itemset and *D* is the size of the database [2]. Support in this approach to action rules is defined like association rules except instead of calculating the support based on a set of transactions; it applies the metric towards the decision table [5]. Pawlak [13] defines the decision table as an information system $S = (U, A_1 \cup A_2 \cup d)$ where *U* is a nonempty finite set, A is a nonempty finite set of attributes, elements of A_1 are stable conditions, elements of A_2 are flexible conditions, U represents the finite set of states (called the universe), and $d \notin A_1 \cup A_2$.

An association rule is defined as $object_x \Rightarrow object_y$ [2]. In the definition, $object_x$ represents the preceding object known as the antecedent and $object_y$ represents the implication of that object known as the consequent. Both association rules and action rules have a similar rule structure. The difference between them is that action rules provide additional information to guide a user and an association rule just provides correlation.

A rare association rule has numerous definitions in the literature, explained in detail in Section 2.4, but will be defined as any rule that is not frequent, or does not meet a minimum support [2, 5]. In action rules this is seen as $support\left(\left[\left[(a, a_p) * (b, b_p \rightarrow b_k)\right] \Rightarrow (d, d_p \rightarrow d_k)\right]\right) \leq support_t$ [5]. In association rules this is seen as

$\frac{count(c)}{database\ size} \leq support_t$, where c is an itemset (set of items) and *t* is a user supplied support [2]. Both definitions refer to any rule that does not meet the support requirement as infrequent or rare.

Action rules and Association Rules are both composed of action sets and itemsets, respectively. An action set was introduced refers to components that make up the action rule [5]. For example, in the rule $[(a, a_1 \rightarrow a_2) * (b, b_1 \rightarrow b_2)] \Rightarrow (d, d_1 \rightarrow d_2)$ it is broken into two action sets: $[(a, a_1 \rightarrow a_2) * (b, b_1 \rightarrow b_2)]$ and $(d, d_1 \rightarrow d_2)$. The same is seen in association rules where an association rule is composed of itemsets on either side of the rule.

2.2 Action Rules from Classification Rules

The notion of an action rule was introduced to address how to increase profit in a company [4]. This research references customers as objects and attributes are features such as bank offers, characteristics, etcetera [4]. Together these objects, attributes, and class form an action rule. A decision table is created based on extracted classification rules [4]. The algorithm pinpoints customer group classifications and shifts flexible attributes, such as an offered promotion, to move a customer from an undesirable group into a desirable group [4].

Semi-stable attributes were introduced by Ras and Tzacheva [14]. This work assumes that not all stable attributes are permanently stable. The research established semi-stable attributes as an attribute that is a function of time [14], such as age. This allows for an extended set of action rules where the potential profit for a company may not be ideal currently but rather than having the ability to change a flexible attribute, time

would act as the change. These action rules are generated firstly through extracting classification rules from a decision table. Once generated, the stable attributes in the action rules that have low confidence are checked for semi stable possibilities and, if applicable, are replaced with the flexible transition. These enhanced rules always have a confidence greater than the rule they have replaced [14].

Tzacheva and Ras [8] introduce a new way to partition the attributes into stable, semi-stable, and flexible. Using these semi-stable attributes, action rules that were once of low interest can be replaced with action rules that have higher potential. The research uses classification rules extracted from a decision table but introduces cost and feasibility of action rules. The cost of an action rule is measured on a scale of $[0, +\infty)$ where 0 is the lowest cost and infinity representing the highest [8]. Each action rule is given a cost. Feasibility is the difference in cost between action rules [8]. If one action has a lower cost than another it is seen to be more feasible to act upon the lower cost [8].

Tsay and Ras propose extended action rules or E-action rules in Discovering Extended Action Rules (System DEAR) [15]. These action rules aim to identify specific transitions that an attribute must undergo to get a desired class transition or class reclassification. Tsay and Ras extend this work in DEAR2 [16]. DEAR2 uses a tree structure to provide a more efficient strategy for rule generation. The tree structure partitions the data by the labels of the stable attributes which reduces the amount of rule comparisons to equivalence classes rather than a comparison of all rules [16]. Additionally, DEAR3 extends DEAR2 by focusing on 3 major improvements [17].

Tzacheva and Ras [18] present a method to generate action rules using only single classification rules. Using a single classification rule reduces the complexity of creating

the action rule and reduces the time complexity from the classical action rule methods. The usage of single classification rules is made possible due to the introduction of the header of an action rule. This header is composed of a rule's stable attributes and assists in predetermining cost and feasibility overhead with respect to the number of stable attributes the header contains. The more attributes that the header contains, the lower the cost and feasibility are. This method allows for user constraints such as feasibility, maximal cost, and minimum confidence to be used as a pruning point for the action rules [18]. This work is an improvement to the work proposed by Tzacheva and Ras [8] by introducing headers of action rules along with a modified A* graph search space [18].

2.3 Action Rules without Classification

Association Action Rules (AAR) steers away from the classical method of action rule generation [5]. Rather than requiring the usage of extracted classification rules, AAR discovers the action rules straight from the decision system [5]. AAR also introduces atomic action sets which are defined as $(a, a_1 \rightarrow a_2)$, where a is an attribute and $a_1, a_2 \in V_a$; where V_a is the domain of a [5]. Retaining the attribute definitions from the classical action rule generation; a is said to be stable if $(a, a_1 \rightarrow a_1)$ and is written at (a, a_1) and a is said to be flexible if $(a, a_1 \rightarrow a_2)\ iff\ a_1 \neq a_2$ [5]. AAR also uses a minimum support threshold to generate frequent or interesting action rules where generated action rules must exceed this threshold. AAR generates its action sets along with a support for each action set. Once all the action sets have been generated it then generates the action rules from the frequent, greater than the set support, action sets.

Action Rule Extraction from Decision table (ARED) is an approach proposed by Im and Ras [19]. ARED extracts action rules without pre-existing classification rules and extracts all distinct action rules that have minimal attribute involvement. This work defines granules or a set of objects that contain a particular attribute. Once these granules have been generated, ARED then checks for possible property transitions between the objects and if the transition is valid, it generates that as the action rule [19].

Utilizing a marking strategy, the Action Rule Discovery (ARD) algorithm generates the action rules directly from the decision system. Proposed by Ras and Dardzinska [9], the ARD algorithm begins with an atomic action term of size 1 and iteratively goes through the action terms checking support, where $support(r) = card(Y_1 \cap Z_1)$, and confidence where $confidence(r) = \left[\frac{card(Y_1 \cap Z_1)}{card(Y_1)}\right] * \left[\frac{card(Y_2 \cap Z_2)}{card(Y_2)}\right]$. Using this method, every term begins unmarked and can then be either marked positive or negative. The iteration continues until all terms have been marked; those that received a positive mark become action rules and those that have received a negative mark are discarded [9].

The proposed Mining Action Rules from Scratch (MARFS) algorithm [6] introduces the notion of PE and NE being positive example and negative example respectively. These objects declare the current class state that an object is in with the goal being to go from $PE \rightarrow NE$ [6]. This method has a spin on the traditional support seen in association mining and in action rule mining such as AAR [5]. MARFS calculates two supports of $NL \rightarrow PL$ where $L = \Lambda_{j \in J}[A_j = v_j]$ as the percentage of NE that satisfy NL under the background of positive PE and as the percentage of PE that satisfy PL under the

background of positive NE [6]. Although MARFS is stated to be inefficient in the literature, it is used as an exploratory algorithm and generates frequent action rules [6].

Difallah, Benton, Raghavan, and Johnsten [11], propose a change from the decision table to a novel action table. This table changes the problem from a classification centric problem to an association mining problem. To improve the computational complexity, the Frequent Association Action Rules Mining (FAARM) algorithm utilizes an efficient algorithm FPGrowth [20] to quickly search through a FP-Tree [20] pruning those action rules that do not meet the support threshold.

Daly, Benton, and Johnsten [7] proposed Multi-Objective Evolutionary Action Rules (MOEAR). MOEAR initially builds action rules by randomly selecting a stable attribute with its value, a flexible attribute with both its initial and final value, and a class with its value. MOEAR then uses a "Sort Log Non-Dominated" method which utilizes Pareto analysis to retrieve a given population of action rules [7]. This work is the first to directly note the occurrence of "rare action rules" and generate them while generating strong action rules, action rules which exceed a support and confidence threshold. Compared against AAR, MOEAR is observed to be immensely faster in execution time [5]. The strong rule coverage between the two are relatively close in the noted experiments except for two data sets of which MOEAR's ability to generate rare action rules closed the gap between the algorithms.

2.4 Rare Association Rules

Intuitively, rare can be identified as the opposite of frequent or it can be an instance that does not meet some threshold. Szathmary, Napoli, and Valtchev introduce a

novel definition of rare in minimal rare itemset [21]. This work utilizes a lattice structure and introduces rare and frequent zones, or the point at which an itemset reaches the threshold between being rare and frequent [21]. A minimal rare itemset is a rare itemset of which all its proper subsets are frequent and a maximal rare itemset if all its proper supersets are rare [21].

Koh and Rountree propose two definitions of rare: perfectly and imperfectly sporadic [22]. Perfectly sporadic rules are association rules of which every item in the rule is rare. Imperfectly sporadic rules are association rules that contain items that are both rare and frequent [22]. Both perfectly and imperfectly sporadic rules are under the sporadic rule search which is defined as any rule that falls below a user set maximum support and above a user set confidence threshold [22]. Although this work proposes both perfectly and imperfectly sporadic rules the focus remains on perfectly sporadic.

Moving away from the level-wise approaches of generating rare association rules, Tsang, Koh, and Dobbie [23] propose a tree structure, "RP-Tree", to allow direct rare association rule generation. This approach introduces a new restraint measure known as minimum rare support, which is the lowest support an item can have to be considered rare. This is used in unison with the standard notion of support to create a threshold with upper and lower bounds as seen as $minimum\ rare\ support \leq support(x) \leq minimum\ support$. In this threshold any itemset that exceeds the upper bounds is frequent and any itemset that has support lower than minimum rare support is of no interest [23]. The RP-Tree algorithm is an adaptation of the FPGrowth algorithm [20]. In both the FPGrowth and RP-Tree algorithm, the database is initially scanned to generate

item support. On the second scan RP-Tree only builds its tree on transactions that contain at least a single rare item [23].

Bayardo Jr., Agrawal, and Gunopulos [24] propose an efficient, consequent constraint rule miner for dense databases in Dense Miner. This approach does is not directly targeted rare rules but limits the rules that are generated to only rules that have the consequent supplied by a user [24]. This applies to rare association mining because of Dense Miner's ability to handle rules with extremely low support [24]. Rare rule generation is achieved by setting the consequent to a known rare consequent and all rules generated would contain on rules with the consequent supplied by the user, in this case the rare consequent [24].

CHAPTER III

METHODOLOGY

The focus of this methodology was the present a definition of rare action rules and lay out the algorithm that will be used to find them. Section 3.1 provides an introduction and foundation, Section 3.2 describes the formal definitions, and Section 3.3 elaborates on the proposed algorithms.

3.1 Introduction

The goal of this research project is to develop a definition of rare action rules, an algorithm to generate these rare action rules, and identify frequently rare attribute transitions. In the literature review, it was observed that the work done in rare action rules is limited. The main algorithm presented in this work is named Rare Action Rule Exploration (RARE). This algorithm takes a consequent constraint approach that uses a support framework inspired by the support boundary framework of RP-Tree [23]. The RARE algorithm will only generate rules that contain a user defined consequent as a starting position while meeting a threshold of support criteria. These rules can be seen going from: $initialState_{rare} \rightarrow finalState_{frequent}, initialState_{frequent} \rightarrow finalState_{rare}, and\ initialState_{rare} \rightarrow finalState_{rare}$ where the state prior to the transition ($\rightarrow$) would be the user defined consequent.

For example, in a nuclear reactor plant a user could supply a class attribute such as reactor state is stable. The rules generated would all contain this class transition (e.g., $Reactor_{state}, stable \rightarrow meltdown$). This allows for rule generation of a given class's state transition or $(d, d_k \rightarrow d_n)$ where *d* is the class, $d_k$is any starting state of the class attribute, and d_n is the resulting state after transition. The motivation for this rule generation method comes the consequent constraint method used by Dense Miner [24].

The generation of the rare action rules via consequent constraint provides a general overview of the system. To assist in hypothesis building, this work also proposes a second algorithm for attribute analysis (AAA) that identifies key actions to investigate based on frequent attribute transitions within the generated rare action rules using a confidence framework. Once the rare action rules are generated, they become our search space where attribute transitions with high confidence can be found.

3.2 Formal Definitions

Before describing the proposed algorithms, some terms must be defined. This thesis uses the definition of an action rule where an action rule is noted as $\left[(b_1, v_1 \rightarrow w_1) * (b_2, v_2 \rightarrow w_2) * \ldots * \left(b_p, v_p \rightarrow w_p\right)\right] \Rightarrow [(d, k_1) \rightarrow (d, k_2)]$, where *b* is the attribute, *v* is the attribute's initial state, *w* is the attribute's changed state (from $v \rightarrow w$), *d* is the class attribute, and *k* is the class attribute's transition [4]. In this action rule definition, anything prior to the implication, seen above as $\left[(b_1, v_1 \rightarrow w_1) * (b_2, v_2 \rightarrow w_2) * \ldots * \left(b_p, v_p \rightarrow w_p\right)\right]$, is known as the antecedent. Anything after the implication, seen above as $[(d, k_1) \rightarrow (d, k_2)]$, is known as the consequent.

The structure used to store the data will be an implementation of a n-ary tree defined as an “Action Tree”. The root of the tree will contain a class attribute value and each proceeding level of the tree will represent a single attribute value from that class value. The height of each tree is *n+1* with *n* being the number of attributes and each node can have up to *k* number of children with *k* being the number of possible variations of the attribute it represents. An example of an action tree is presented in Figure 1. Class represents a single class value (e.g., *Class*: [A, B]) and attribute represents the attributes corresponding with that class where $Attribute_{1a}$ is the first attribute (1) in *n* number of states (a). To further explain, if given a list of attributes [*a, b, c*] where *a* can be [1, 2, 3], *b* can be [1, 2], and *c* can be [1, 2, 3] then $Attribute_{1a}$ would represent the first attribute (a) value (1) and $Attribute_{1b}$ would represent the first attribute (a) value (2).

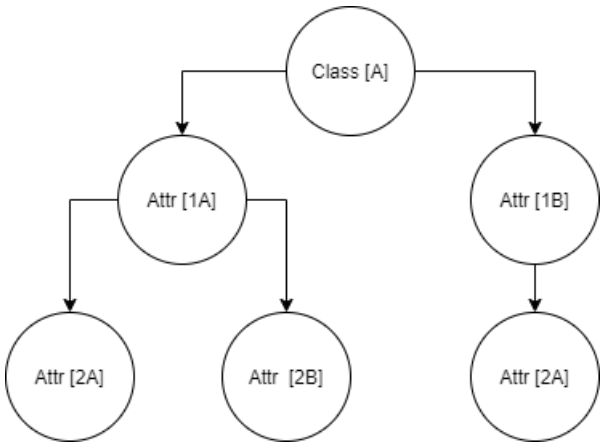

Figure 1. Action Tree Base Model.

The support framework utilizes the definition where support is seen as $\frac{count(x)}{D}$ [2], where *x* is the count of a class value and *D* is the size of the database. This is to see how many times the attribute occurred in the database. Support will be used in a range where the maximum and minimum support thresholds are user supplied and must meet the criteria $0 \leq support_{minimum} \leq 1$, $0 \leq support_{maximum} \leq 1$, and $support_{minimum} < support_{maximum}$. This support range is used to eliminate noise or anomalous rules and to ensure that the algorithm can handle the three consequent transitions: $frequent \rightarrow rare$ and $rare \rightarrow rare$ which require the usage of low support as well as $rare \rightarrow frequent$ which requires high support. Although the definition is adopted, the application of these metrics is altered and will be explained in the following sections.

A rare action rule is an action rule that is generated and contains the user defined consequent as a transition while also being greater than or equal to the *minRareSup* and less than or equal to the *maxRareSup*. Formally an action rule $\left[(b_1, v_1 \rightarrow w_1) * (b_2, v_2 \rightarrow w_2) * \ldots * \left(b_p, v_p \rightarrow w_p\right)\right] \Rightarrow (d, k_u \rightarrow k_n)$ is said to be rare

$$iff \begin{cases} k_u = user\ supplied\ consequent \\ k_n \neq k_u \\ minRareSup \leq support(k_n) \leq maxRareSup \end{cases}$$ where $support = \frac{count(class_{value})}{Database}$.

An example is $[(a, a_1 \rightarrow a_2) * (b, b_1 \rightarrow b_2)] \Rightarrow (d, d_1 \rightarrow d_2)$ where $user\_consequent$ is d_1 and the transition of $(d, d_1 \rightarrow d_2)$ occurs if and only if $minRareSup \leq support_{d_2} \leq maxRareSup$.

Finally, an attribute is said to be a high confidence attribute transition if its confidence is greater than a user supplied confidence where $confidence =$

$\frac{count(Decision_{transition} \cup Attribute_{transition})}{count(Decision_{transition})}$. For example, assume two action rules $[(A1, a \rightarrow b) * (A2, c \rightarrow a)] \Rightarrow (D, no \rightarrow yes)$ and $[(A1, a \rightarrow b) * (A2, c \rightarrow b)] \Rightarrow (D, no \rightarrow yes)$ where both rules are seen to have a class transition $no \rightarrow yes$. With respect to this transition, it is shown that the confidence of the attribute transitions $(A1, a \rightarrow b)$ is 100%, $(A2, c \rightarrow a)$ is 50%, and $(A2, c \rightarrow b)$ is 50%. In this example, if a user was interesting in attribute transitions with a confidence at minimum 60% then $(A1, a \rightarrow b)$ would be of interest to the user and the rule set.

3.3 Proposed Algorithms

There are two proposed algorithms: RARE discovers rare action rules utilizing a consequent-constraint and support framework and the AAA algorithm takes advantage of the confidence framework to generate high confidence rare attribute transitions. A full system workflow is detailed in Figure 2 and is explained throughout the upcoming sections.

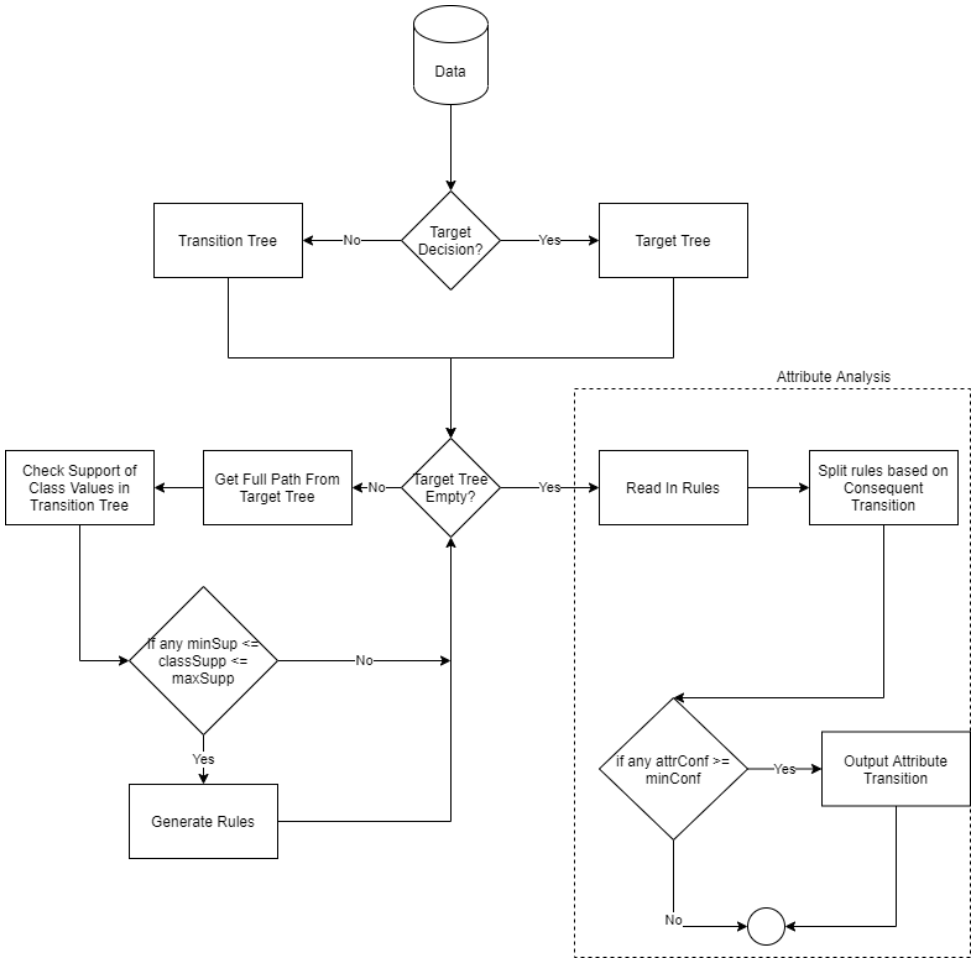

Figure 2. Workflow.

3.3.1 Rare Action Rule Exploration

The first algorithm and primary research contribution is the Rare Action Rule Exploration (RARE) rule generation algorithm. This approach utilizes a consequent-constraint approach which assumes that any transition of the consequent from the user defined consequent is considered rare. The RARE algorithm consist of three major stages are: data partitioning, rule generation, and result output, as demonstrated in Figure 3.

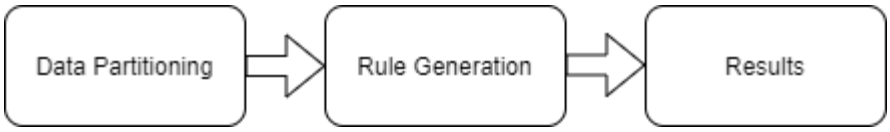

Figure 3. RARE Major Steps.

The data partitioning step requires a user defined class attribute as input which is the target class value, a minimum support, and a maximum support. With user defined class the data is partitioned into two Action Trees; one which contains the user defined class value and all its corresponding attributes from a transaction and another which contains all the non-target class values along with their associative attributes. The target Action Tree will have at most a height of *n+1* where n is the number of attributes and the 1 represents the target class attribute and the non-target Action Tree will have a height of *n+2* where *n* represents the number of attributes and the 2 accounts for the base root level and the level with the non-target class attributes. For example, given the first data entry in Table 1, $[1,\ 2,\ 3,\ A]$, assuming the target class value is A, the tree containing this data would have a height of $n + 1$, which is a height of 4 in this case, where *n* is the number of flexible and stable attributes and the additional 1 account for the class value. This is in Figure 4. Now take the second data entry in Table 1, [2, 1, 3, B], assuming that this is not the target class value it would be placed in the Non-Target Action tree. The Non-Target Action Tree has a height of $n + 2$, in this case the height of the tree would be 5, where *n* is the number of stable and flexible attributes and the additional 2 accounts for the class value and the empty root node. This is seen in Figure 5.

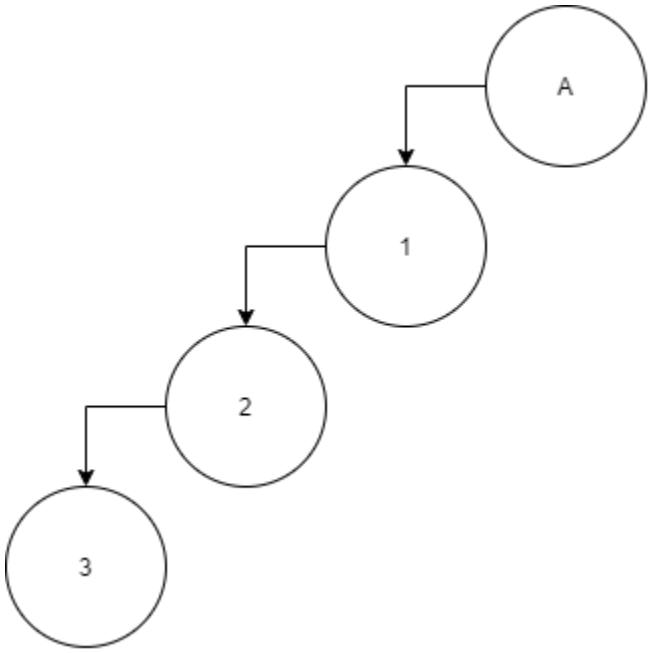

Figure 4. Target Action Tree Height.

Once the data has been properly partitioned, RARE begins to generate rules. This is done using tree comparison between the target and the non-target tree. This process begins by checking for a full path from the root to a leaf, within the target tree. Once that path has been identified there are two rule generation mechanisms that must be considered: (1) the user can give a target class as input and every combination of action rules would be generated (including rare and frequent) and (2) given a target, all the transitions that adhere to the support framework where $minRareSup \leq support_{decision} \leq maxRareSup$ would be identified.

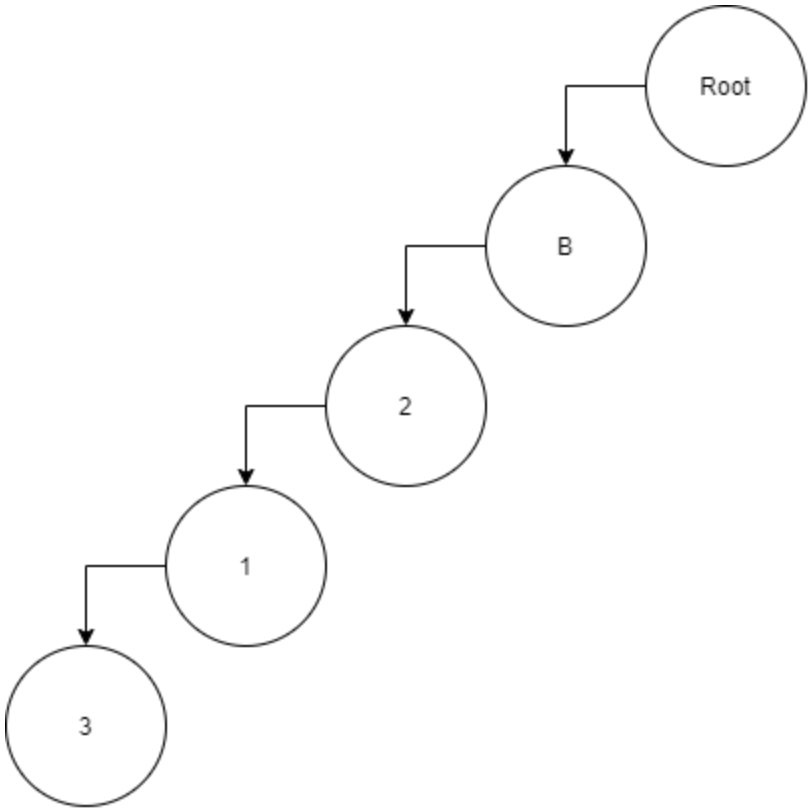

Figure 5. Non-Target Action Tree Height.

If a class value's support does not meet the support range, that branch of the tree will be pruned off. Table 1 shows an example dataset of size 8 (or 8 data entries) where a class target A would be split into 2 trees representing one target Action Tree and another non-target Action Tree.

Starting with the class value, the tree is built level by level by adding the new value or extending the branch if the value exists. The target Action Tree contains two data entries total and has a count of two (all containing class A) and is detailed in Figure 6. The target action tree gets initialized with the first instance of the target class value. From there is begins checking if a node for the first attribute value exists. If it does not

exist it creates a node for that value and then moves to the next attribute value, which becomes the next child in the tree.

Table 1. Example Dataset

Attribute 1	Attribute 2	Attribute 3	Class
1	2	3	A
2	1	3	B
1	2	1	C
1	1	1	C
1	1	2	A
1	1	2	B
2	2	1	C
2	1	2	C

If the values do exist, then no node is created, and the next attribute value is retrieved. The non-target Action Tree containing class B (which has a count of two) and class C (which has a count of four) is shown in Figure 7 and has a total count of six. This tree is constructed in a similar manner except it must check for the creation of class values, as this tree contains all class values that are not the target.

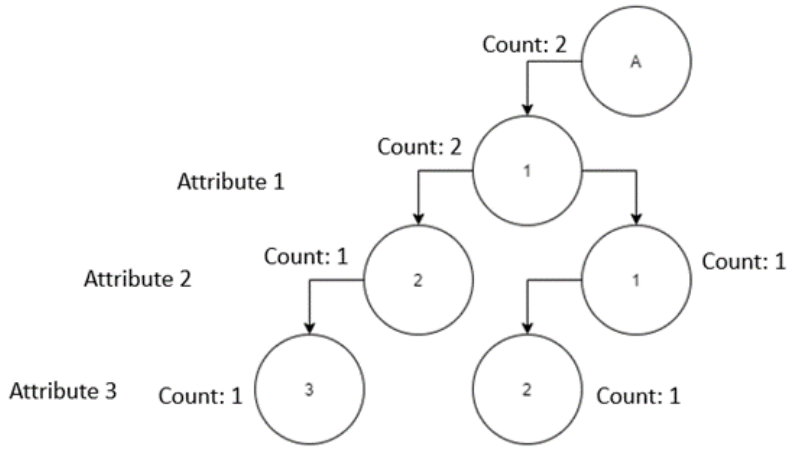

Figure 6. Example Target Action Tree.

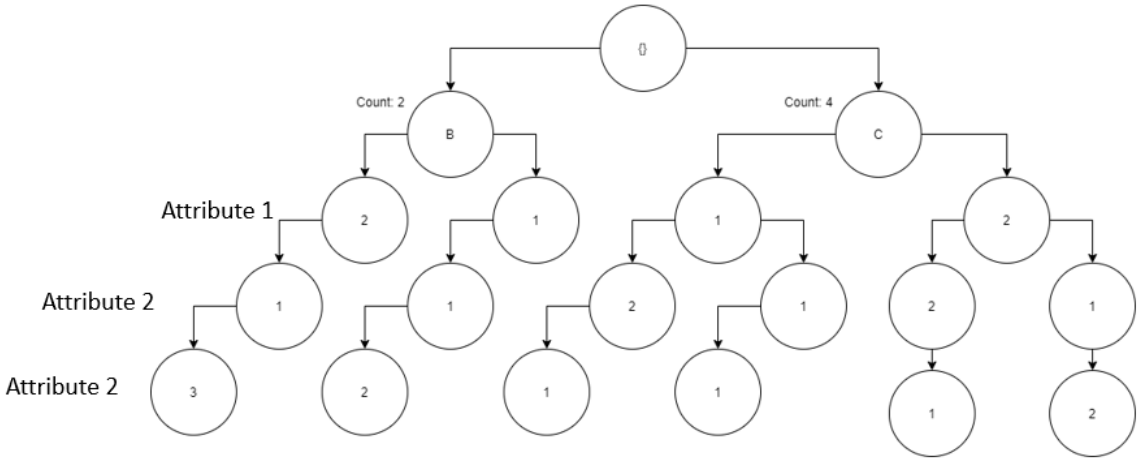

Figure 7. Example Non-Target Action Tree.

As mentioned earlier, there are two different rule generation mechanisms: (1) the user can give a target class as input and every combination of action rules would be generated (including rare and frequent) and (2) given a target, all the transitions that adhere to the support framework where $minRareSup \leq support_{decision} \leq$

$maxRareSup$ would be identified. In the first rule generation mechanism, the rules generated would come from the transitions of all paths in the Target Action Tree to the non-target tree regardless of the support the other classes hold. This generates significantly more rules because this would check a target data entry against every available path in the non-target Action Tree. In the second mechanism, support value thresholds would be in place, which would reduce the number of rules generates and make the rules more relevant to the user. For this example, we set the minimum support to 0 and the maximum support to 0.4 or 40%. This would check the support of the non-target action tree class nodes against the support threshold.

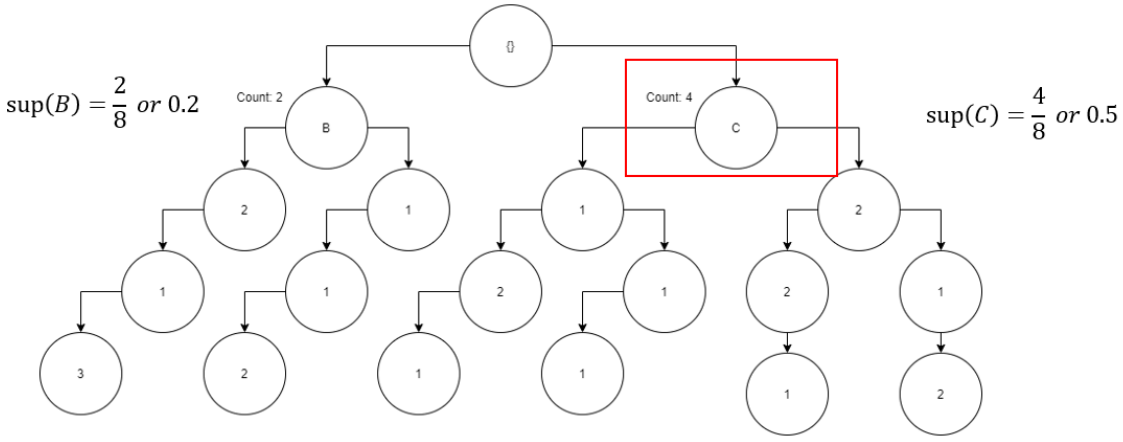

Figure 8. Non-Target Action Tree Support Shown.

Shown in Figure 8, class value B has a support of 0.2 and class value C has a support of 0.5. Generating rules this way would ignore any class values that have a count outside of the support range, in this example class C would be pruned, see Figure 9.

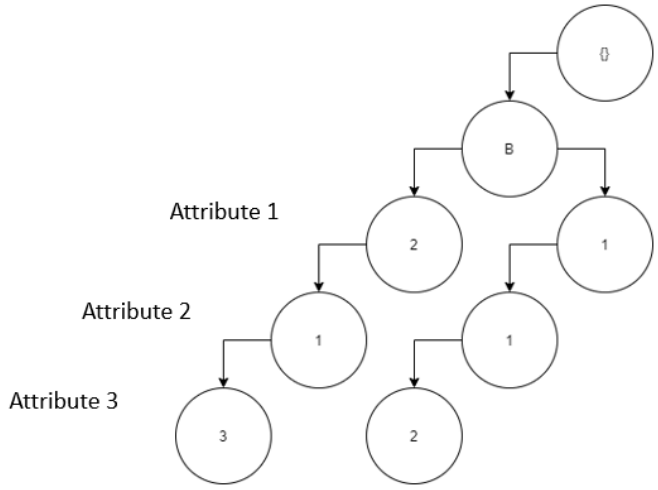

Figure 9. Post Pruning Non-Target Action Tree.

Rules would only be generated from the target Action Tree → Class value B of the non-target Action Tree, reducing the number of rules generated, reducing the search space by 66%, and ensuring that the rules are rare. This example would generate 3 rules: $[(Attr_1, 1 \rightarrow 2), (Attr_2, 2 \rightarrow 1)(Attr_3, 3)] \Rightarrow (d, A \rightarrow B)$, $[(Attr_1, 1), (Attr_2, 2 \rightarrow 1), (Attr_3, 3 \rightarrow 2)] \Rightarrow (d, A \rightarrow B)$, and $[(Attr_1, 1 \rightarrow 2), (Attr_2, 1), (Attr_3, 2 \rightarrow 3)] \Rightarrow (d, A \rightarrow B)$.

Without the support threshold in place, the rules generated would contain both frequent and rare action rules. This work uses the second mechanism to focus on rare action rules. The pseudocode is presented in Figure 10.

```
Step 1 := Data Preprocess
For each data instance:
        If class value == target class:
                add data instance to Target Action Tree
        Else:
                add data entry to Non-Target Action Tree with incrementing class count

Step 2 := Rule Generation
If minSup <= class support (non-target) <= maxSup:
        prune that branch of the tree off

For each data entry in the Target Action Tree:
        For each data entry in Non-Target Action Tree:
                if attribute is flexible:
                        if value changed:
                                record the change
                        else:
                                record the unchanged value
                if attribute is stable:
                        if value changed:
                                drop attribute
                        else:
                                record unchanged value
```

Figure 10. RARE Pseudocode.

3.3.2 High Confidence Attribute Transitions

The AAA algorithm proposed in this work is an attribute analysis algorithm that identifies high confidence attribute transitions. The identified attribute transitions would suggest attributes that hold a higher correlation with a class transition. For large rules this could alleviate the uncertainty of what specifically has a higher correlation with the class transitions. Using the rare action rules generated from the consequent constraint algorithm as input, this algorithm takes the attribute transitions with respect to a class value transition, in this case $(d, A \rightarrow B)$ seen in Table 2.

The algorithm checks the confidence against the user confidence threshold for all attribute transitions. An example confidence threshold of 60% would generate the following attribute transition from Table 2: when $(d, A \rightarrow B)$ then $(A_1, 1 \rightarrow 2)$ and $(A_1, 2 \rightarrow 1)$ with 66% confidence as both transitions occurred twice out of the three class

transitions. This information provides the user suggestive grounds to form stronger hypothesis on what could be directly correlated with the rare class value transition. The pseudocode is shown in Figure 11.

Table 2. Attribute Transition List ($\mathrm{d}_1 \rightarrow \mathrm{d}_2$)

Attribute Transition	Occurrence Count	Total Decision Transition	Confidence
$\mathrm{A}_1, 1 \rightarrow 2$	2	3	66%
$A_2, 2 \rightarrow 1$	2	3	66%
$A_3, 3 \rightarrow 2$	1	3	33%
$A_3, 2 \rightarrow 3$	1	3	33%

The algorithm checks the confidence against the user confidence threshold for all attribute transitions. An example confidence threshold of 60% would generate the following attribute transition from Table 2: when $(d, A \rightarrow B)$ then $(A_1, 1 \rightarrow 2)$ and $(A_1, 2 \rightarrow 1)$ with 66% confidence as both transitions occurred twice out of the three class transitions. This information provides the user suggestive grounds to form stronger hypothesis on what could be directly correlated with the rare class value transition. The pseudocode is shown in Figure 11.

```
Step 1 := Data Preprocessing
Read in the action rules
For each action rule:
        seperate by consequent transition
        seperate transitions of the rule with a count

Step 2 := Attirbute Analysis
For each consequent transition:
        For each attribute transition:
                if attrConf >= minConf:
                        generate high confidence attribute transition
                else:
                        ignore
```

Figure 11. Attribute Analysis Pseudocode.

In Chapter III we have presented a definition for a rare action rule and presented an algorithm to mine these rare action rules. A definition for high confidence attribute transitions and an algorithm to find them have also been described. Chapter IV discusses the experimental setup and Chapter V discusses the results and conclusions from the experiments.

CHAPTER IV

EXPERIMENTAL DESIGN

Experiments were designed to evaluate the validity and scalability of the RARE algorithm and to see how it compared with modified versions of the FAARM [11] algorithm. These experiments were used to determine that the results of the algorithms are accurate according to what was expected, the scalability of the algorithms, and how the algorithms compared to each other.

To reduce inconsistency, all experiments were executed on a Lenovo ThinkStation Windows 10 Pro version 1803 operating system with 32 GB of RAM and an Intel® Xeon® CPU E5-2620 v4 at 2.10GHz processor. All the algorithms were implemented in Java version 1.8.0_161 through the Eclipse Oxygen.2 Release (4.2.2) IDE.

4.1 Modified FAARM

The FAARM algorithm was chosen as the comparison algorithm against RARE. This algorithm was chosen due to its utilization of the action table which closely resembles the process RARE uses with Action Trees. To compare the algorithms on an even scale, some modifications were made to FAARM.

The first modification to the FAARM algorithm targets the attribute-based support framework within the action table while the RARE algorithm uses a transactional based support framework. Altering this key component of FAARM can disrupt the integrity of the algorithm, thus a work around was developed. The support modification to FAARM included adding an upper layer of support to take in a support range identical to the RARE algorithm. This allows the pruning off any consequent that does not meet this support range and is utilized before the action tables are built. This can be seen in the pseudocode shown in Figure 12 where the text in red represents the modifications made while the rest remain true to the pseudocode seen in the original FAARM algorithm [11].

```
For each class value:
    If minRareSupport ≤ support(class) ≤ maxRareSupport:
        Generate all the atomic action sets from A.
        Calculate the frequency of each atomic set.
        Build the action table.
        Prune and reorder the action table.
        Build the FP-Tree from the action table.
        Run FP-Growth on the FP-tree.
            Extract qualifying action sets
            Build and test the action rules.
```

Figure 12. Modified FAARM Pseudocode.

The second aspect of FAARM that was modified was its' target consequent parameters which required a starting consequent and an ending consequent. Removing the ending consequent allows FAARM to look for all transitions of the consequent that adhere to the new, additional level of support.

These two modifications are used together to prune off the data instances that do not contain a consequent transition that meet the new requirements. Once this process has concluded, FAARM can begin generating rules using its own attribute-based support technique. To generate rules for every transition that makes it through the initial constraints, FAARM builds independent action tables per transition and generates rules from each action table.

Table 3. Data Complexity Comparison

Data Set	Instances	Num of Class Values	Num of Attr.	Max Attr Variants	Min Attr Variants
Balance*	625	3	4	5	5
Breast Cancer	286	2	8	13	2
Car	1728	4	5	4	3
Dermatology*	367	6	32	10	4
Diabetes	520	2	19	2	2
Hepatitis*	155	2	19	9	2
Lung Cancer*	28	3	55	4	4
Nursery	12960	5	7	5	2
Teacher Evaluation*	151	3	4	27	2
Tic Tac*	958	2	8	3	3

4.2 Scalability Experiments

These experiments consisted of evaluating the algorithms against data of different complexity (the number of attributes, class values, and the variations of both) and density (the amount of data instances). A total of 10 data sets that range in complexity and density, were used and can be seen in Table 3. Due to the way the original FAARM code handled class values, there were FAARM specific versions that have a duplicated class value (e.g., "w" is now "ww"). These modified data sets are indicated with an asterisk (*) in Table 3 and include: "Balance", "Dermatology", "Hepatitis", "Lung Cancer", "Teacher Eval", and "Tic Tac".

4.3 Algorithm Comparison Experiments

After both the RARE and FAARM algorithm went through the validity and scalability experiments, the results were compared. These results analyzed the performance (measured in milliseconds and converted to minutes if large enough), memory usage of the program (measured in Megabytes and Bytes), and rule count (measured by total count). The comparisons can be used to show if one algorithm performed better in a specific scenario or if one algorithm performed better overall.

For every data set the input parameters for both the RARE and FAARM algorithms had to be adjusted. This was due to the difference in size of the data ranging from 28 – 12,960. Both RARE support and FAARM's new layer of support were set to the lowest known percentage in the data. For example, if a data set had a class value that occurred 20% of the time, then the support would be set to *minRareSup = 0* and *maxRareSup = 0.2*. For FAARM's inner support it would be set to the raw number of

occurrences. For example, if a dataset had 100 instances and a class value occurred 10 times then the inner support of FAARM would be set to 10.

4.4 Datasets

The data sets used in these experiments were Balance, Breast Cancer, Car Evaluation, Dermatology, Diabetes, Hepatitis, Lung Cancer, Nursery, Teaching Assistant Evaluation, and Tic-Tac-Toe Endgame. All these data sets are publicly available and were be found on the UCI Machine Learning Repository website [25]. A breakdown of the datasets is shown in Table 3 where: *Data Set* is the name of the data, *Instances* is the number of entries (size of the database), *Num of Class Values* is the number of possible values for the class value to be, *Num of Attr* is the number of attributes the data has, *Max Attr Variants* is the max possible number of values for a single attribute to have, and *Min Attr Variants* is the smallest amount of values an attribute can have.

CHAPTER V

RESULTS AND CONCLUSIONS

The experiments set out to analyze how the RARE algorithm compared to a modified FAARM algorithm in memory usage, runtime, and rule generation across data of different size and complexity. Additionally, in these experiments, FAARM was run in two different modes with one mode only using the newly added level of support and another mode using both the new support and the native FAARM support. The first mode created a version that most closely matched the RARE algorithm behavior, but at the potential cost of generating many more rules. FAARM generated all valid rules whereas RARE tended to generated only rules of maximal length. The second mode of FAARM pruned the rules by applying its definition of support, which should result in lower run time than the first mode as well as fewer rules.

Each algorithm searched for $frequent \rightarrow rare$ transitions (FR), $rare \rightarrow frequent$ transitions (RF), and $rare \rightarrow rare$ transitions (RR). This abbreviation is added to the end of the run's name. The three algorithms are denoted as follows: RARE, FAARM_S (FAARM using both levels of support), and FAARM_nS (FAARM using only the newly added support).

The results have been split into two groups that are based on the complexity of the data. The groups contain data sets that fall in either *Low Complexity*, where the number

of attributes range from 4 – 8, or *High Complexity*, where the number of attributes range from 19 – 55. A maximum run time of 3 hours was imposed for a single run. Three hours was identified as the cut off time due to the relatively small size of the datasets and the assumption that they all would complete within this time.

5.1 Memory Usage Results

Memory was measured from the program in both Megabytes and Bytes. The results were then averaged together within each run of a particular algorithm. Results for the low complexity data, show all three algorithms met completion in 5 out of 6 of the experiments; however, in the Tic Tac experiment the FAARM_ns had to be stopped when it exceeded the allotted time of 3 hours.

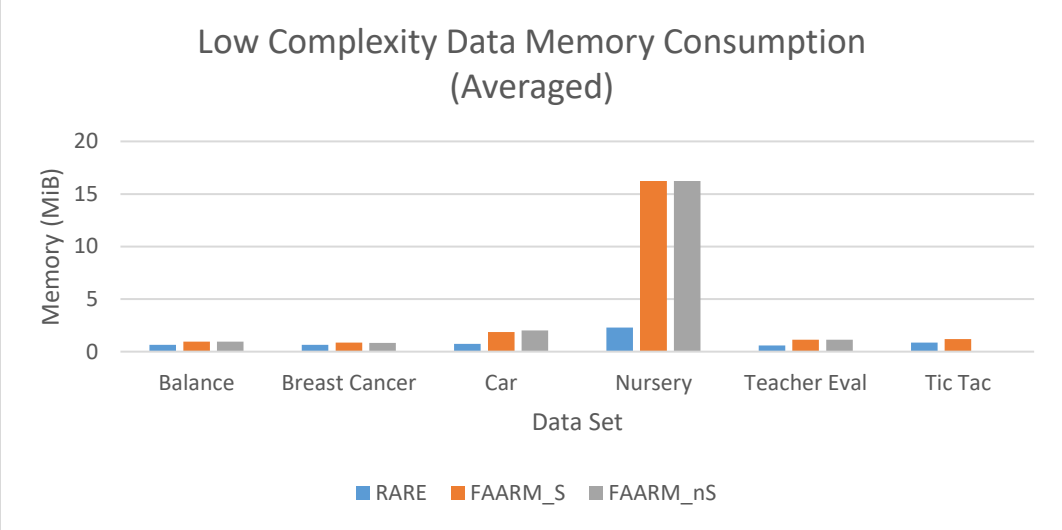

Figure 13. Low Complexity Memory Consumption.

As seen in Figure 13, the RARE algorithm uses less memory than both instances of the FAARM algorithm for all the low complexity datasets. This is likely due to the design and implementation of the modified FAARM algorithm. Each consequent that persists through the pruning stage has an individual action table constructed, while the RARE algorithm generates only two action trees, one of which undergoes pruning immediately.

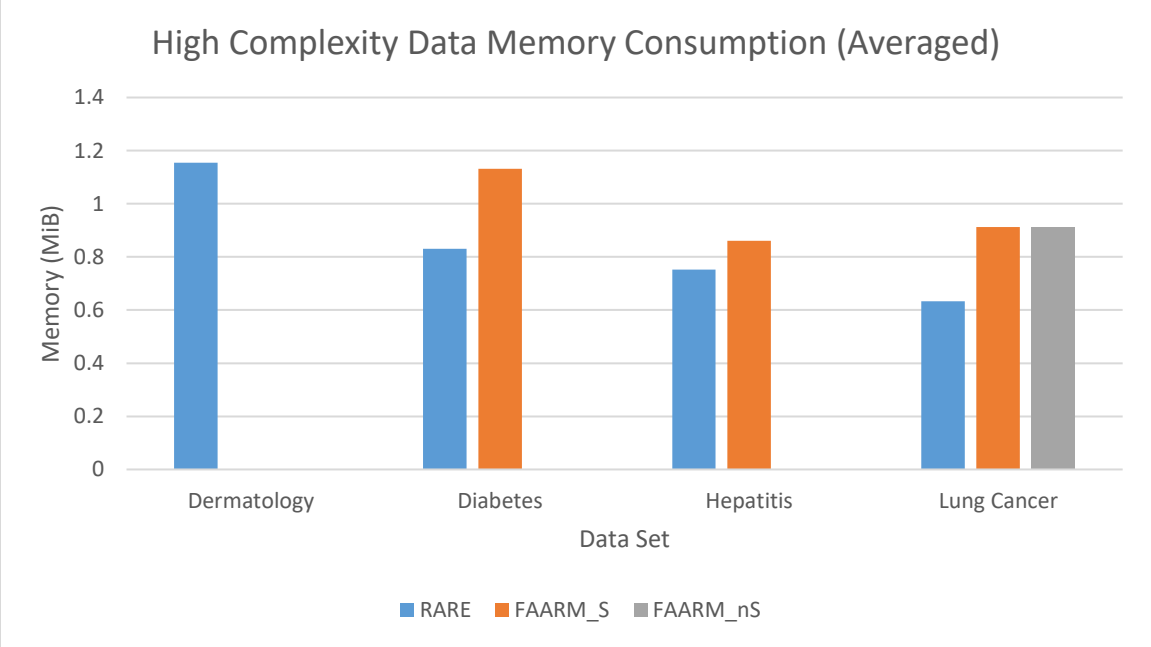

Figure 14. High Complexity Memory Consumption.

When applied to the high complexity data, the RARE algorithm also utilizes less memory, shown in Figure 14. The experiments that used the high complexity data sets only had a single experiment with all the algorithms meeting completion, the Lung

Cancer data set. The FAARM_nS algorithm completed only 1 out of 4 experiments and the FAARM_S algorithm completed 3 out of 4 of the experiments. Unlike in the low complexity experiments, the lack of completion was due to a combination of running out of memory and being time terminated. This was expected based of FAARM's attribute-based rule generation when applied to high complexity data sets.

5.2 Runtime Results

The runtime for the experiments were measured in milliseconds and converted into seconds. The results from each experiment were averaged with other experiments with the same data per algorithm.

In the low complexity data experiments, both RARE and FAARM _S completed all the experiments while FAARM_nS completed 5 out of 6 of the experiments due to time termination. As shown in Figure 15, the FAARM_S algorithm finished execution with greater speed than the RARE algorithm across all low complexity experiments. A particular experiment to note is the Tic Tac experiment. In this data set the RARE algorithm experiences its worst-case situation where the data has a binary class value. The RARE algorithm does not scale well with binary class values and increased data instances. This binary class value keeps RARE from utilizing its tree pruning method and must compare every path of the Target Tree to that of the Non-Target Tree.

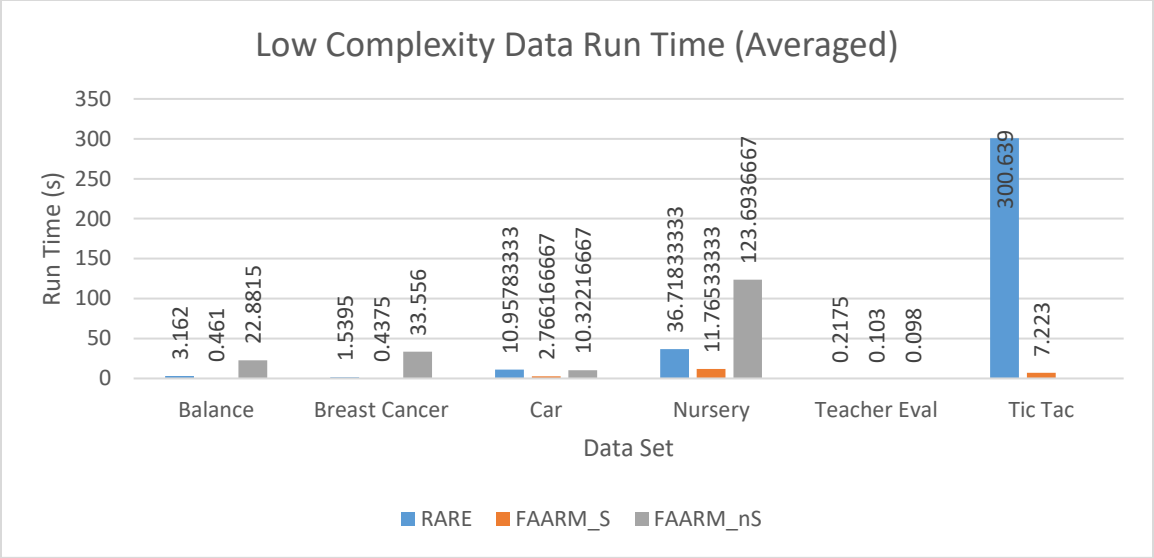

Figure 15. Low Complexity Run Time.

In the high complexity data experiments, the FAARM_S algorithm completed 3 out of 4, the FAARM_nS completed 1 out of 4, and RARE completed 4 out of 4 experiments, shown in Figure 16. Comparing the results of the experiments that had a form of the FAARM algorithm complete, it is seen that the RARE algorithm's execution time exceeds that of the FAARM_S and FAARM_nS. The Diabetes data set shows a drastic improvement of runtime. This is likely due to the number of attributes that the FAARM algorithm had to examine when generating rules.

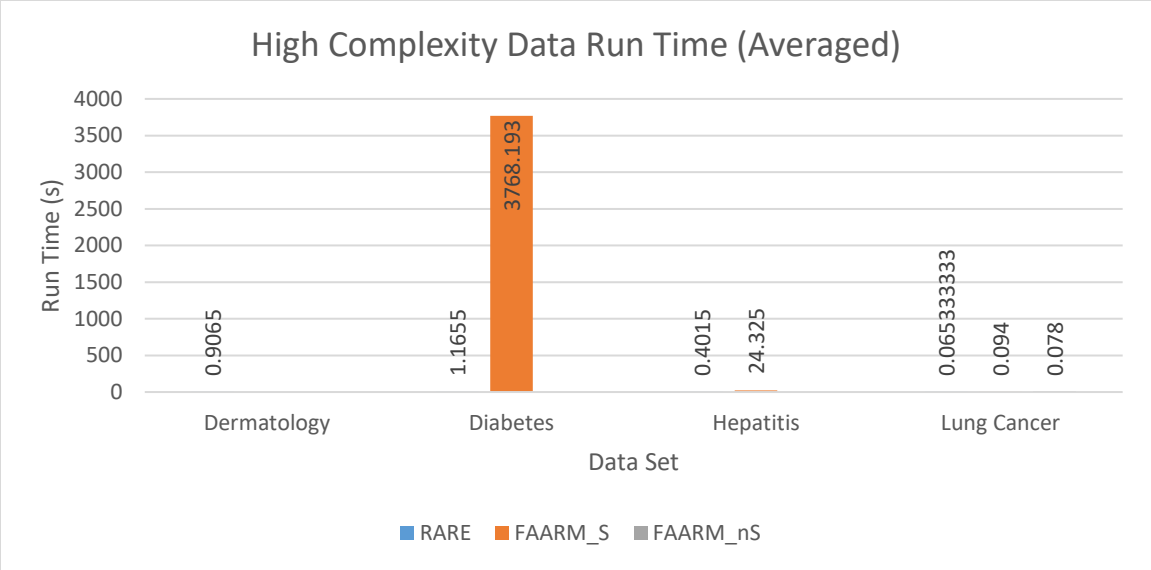

Figure 16. High Complexity Run Time.

5.3 Number of Rules Results

The number of rules generated by the RARE, FAARM_S, and FAARM_nS algorithms varied greatly. In the low complexity data, the FAARM_S algorithm generated the least number of rules, followed by RARE with the middle according to count, and FAARM_nS with the most rules. However, that is not the case in the high complexity data experiments. With high complexity data, the RARE algorithm generated less rules than FAARM_S. In the instance that FAARM_nS completed, it failed to generate any rules. The number of rules generated per algorithm can be seen in Table 4. This variation is due to RARE generating maximal length rules, where the shortest a rule can be is the number of flexible attributes and the FAARM algorithm generating rules of size *n* where a size of 1 is the atomic action set. The results shown in Table 4 and Table 5

should be viewed based on whether a user want to identify all possible rules or just maximal length rules.

Table 4. Low Complexity Number of Rules

Number of Rules						
Low Complexity	Balance	Breast Cancer	Car	Nursery	Teacher Eval	Tic Tac
RARE	21168	14072	12542.67	9063	1751	207832
FAARM_S	407	936	1554.333	2225.333333	0	15458
FAARM_nS	64328.5	74431.5	10322.17	23868	0	null

Table 5. High Complexity Number of Rules.

Number of Rules				
High Complexity	Dermatology	Diabetes	Hepatitis	Lung Cancer
RARE	6293	8839	3788	77
FAARM_S	null	616952	60947	0
FAARM_nS	null	null	null	0

5.4 High Confidence Attribute Results

Finally, the experiments for the AAA algorithm all failed. The algorithm was tested on all result sets generate from RARE experiments and never completed any. This is likely due to the design of the algorithm which has severe scalability problems. In its current state, the algorithm is not usable for practical scenarios.

5.5 Significance of Work

The outcome from the rare action rules shows the changes that have a high correlation with a class value's change. This could be especially beneficial in systems where a class is a reactor's state (Stable, Unstable, Meltdown) and there was interest in any transition of the reactor from Stable to another state, such a nuclear power plant. Although this is a less an unlikely occurrence, the significance of this algorithm can show an actionable correlation given a class state change.

The outcome from the attribute analysis algorithm can further identify high confidence attribute transitions within all the action rules of a particular class state change. For example, if a nuclear reactor changed from stable to meltdown and 80% of those transitions had a very specific attribute of the consistent transition this would indicate a high confidence attribute transition. This algorithm was built to distinguish the attributes in the case that a system has multiple attributes where the transition may not be directly apparent and to allow for a highly confident correlation-based hypothesis to be constructed.

5.6 Conclusions

Although rarity has been greatly explored in association mining, there has been little to no work done with rare action rules. To address the lack of work done, a new definition of a rare action rule and an algorithm to generate these rare action rules is proposed using the RARE algorithm. This method provides a way to generate consequent-constraint rare action rules through simple action tree comparisons. This algorithm has been shown to be more efficient in high data complexity scenarios than two

modified versions of FAARM, one using FAARMS native support and another without. The modified FAARM algorithms created action rules that met the proposed rare action rule definition.

In datasets with a smaller number of attributes, RARE has been shown to require more time and generates more rules than the FAARM algorithm with native support, while still requiring less memory. As a result, the RARE algorithm would be a preferable algorithm to run for rare rule generation.

5.7 Future Work

In terms of future work, the current implementation of the RARE algorithm and the attribute analysis algorithm can be improved. At the current time, RARE generates only maximal action rules where the shortest an action rule can be is the number of flexible attributes within the data. Adapting this implementation to be able to generate action rules of all sizes could prove to be beneficial. Another goal would be to improve the efficiency of the algorithm when dealing with low complexity data. This could be handled through implementing an alternate method of tree traversal such as FP-Tree and FPGrowth algorithm approach. Another implementation adaption could be through tree traversal. The attribute analysis could be improved through the complete efficiency or an algorithm improvement. Another area to be examined is to look for alternate definitions for a rare action rule created. Additionally, alternative algorithms to handle the presented definition may be able to be developed.

REFERENCES

REFERENCES

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

[2] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, 1993, pp. 207-216.

[3] J. Han, J. Pei, and M. Kamber, *Data mining: Concepts and Techniques*. Elsevier, 2011.

[4] 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*, 2000, pp. 587-592: Springer.

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

[6] Z. He, X. Xu, S. Deng, and R. Ma, "Mining action rules from scratch," *Expert Systems with Applications*, vol. 29, no. 3, pp. 691-699, 2005.

[7] G. Daly, R. Benton, and T. Johnsten, "A Multi-Objective Evolutionary Action Rule Mining Method," in *2018 IEEE Congress on Evolutionary Computation (CEC)*, 2018, pp. 1-8: IEEE.

[8] A. A. Tzacheva and Z. W. Raś, "Action rules mining," *International Journal of Intelligent Systems*, vol. 20, no. 7, pp. 719-736, 2005.

[9] Z. W. Raś and A. Dardzińska, "Action rules discovery without pre-existing classification rules," in *International Conference on Rough Sets and Current Trends in Computing*, 2008, pp. 181-190: Springer.

[10] S. Im and Z. W. Raś, "Action rule extraction from a decision table: ARED," in *International Symposium on Methodologies for Intelligent Systems*, 2008, pp. 160-168: Springer.

[11] D. E. Difallah, R. G. Benton, V. Raghavan, and T. Johnsten, "FAARM: Frequent association action rules mining using FP-Tree," in *2011 IEEE 11th International Conference on Data Mining Workshops*, 2011, pp. 398-404: IEEE.

[12] P. Hájek, I. Havel, and M. Chytil, "The GUHA method of automatic hypotheses determination," *Computing*, vol. 1, no. 4, pp. 293-308, 1966.

[13] Z. Pawlak, "Rough sets and decision tables," in *Symposium on Computation Theory*, 1984, pp. 187-196: Springer.

[14] Z. W. Ras and A. A. Tzacheva, "Discovering semantic inconsistencies to improve action rules mining," in *Intelligent Information Processing and Web Mining*: Springer, 2003, pp. 301-310.

[15] Z. W. Ras and L.-S. Tsay, "Discovering extended action-rules (System DEAR)," in *Intelligent Information Processing and Web Mining*: Springer, 2003, pp. 293-300.

[16] L.-S. Tsay* and Z. W. Raś, "Action rules discovery: system DEAR2, method and experiments," *Journal of Experimental & Theoretical Artificial Intelligence,* vol. 17, no. 1-2, pp. 119-128, 2005.

[17] L.-S. Tsay and Z. W. Raś, "Action rules discovery system DEAR_3," in *International Symposium on Methodologies for Intelligent Systems*, 2006, pp. 483-492: Springer.

[18] A. Tzacheva and Z. W. Raś, "Constraint based action rule discovery with single classification rules," in *International Workshop on Rough Sets, Fuzzy Sets, Data Mining, and Granular-Soft Computing*, 2007, pp. 322-329: Springer.

[19] S. Im and Z. W. Raś, "Action rule extraction from a decision table: ARED," in *International Symposium on Methodologies for Intelligent Systems*, 2008, pp. 160-168: Springer.

[20] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," *ACM Sigmod Record,* vol. 29, no. 2, pp. 1-12, 2000.

[21] L. Szathmary, A. Napoli, and P. Valtchev, "Towards rare itemset mining," in *19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007)*, 2007, vol. 1, pp. 305-312: IEEE.

[22] Y. S. Koh and N. Rountree, "Finding sporadic rules using apriori-inverse," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2005, pp. 97-106: Springer.

[23] S. Tsang, Y. S. Koh, and G. Dobbie, "RP-Tree: rare pattern tree mining," in *International Conference on Data Warehousing and Knowledge Discovery*, 2011, pp. 277-288: Springer.

[24] R. J. Bayardo, R. Agrawal, and D. Gunopulos, "Constraint-based rule mining in large, dense databases," *Data Mining and Knowledge Discovery,* vol. 4, no. 2-3, pp. 217-240, 2000.

[25] https://archive.ics.uci.edu/ml/datasets

BIOGRAPHICAL SKETCH

BIOGRAPHICAL SKETCH

Name of Author: Blake A. Johns

Graduate and Undergraduate Schools Attended:

The University of South Alabama, Mobile, Alabama

Degrees Awarded:

Masters of Science in Computer and Information Sciences, 2021. Mobile, Alabama

Bachelors of Science in Computer Science, 2020, Mobile, Alabama

Awards and Honors:

Graduate Research Assistant, School of Computing, Summer 2020 – Fall 2021

Undergraduate Researcher, School of Computing, Spring 2019 – Spring 2020

Computing Undergraduate Research Experience, School of Computing, Fall 2018