MR-Apriori Count Distribution Algorithm for Parallel Action Rules Discovery

Angelina A. Tzacheva, Midhun M. Sunny, and Pranava Mummoju
Department of Computer Science
University of North Carolina at Charlotte
Charlotte, NC, USA
E-mail: {aatzache, msunny1, pmummoju}@uncc.edu

Abstract—Data mining deals with the extraction of hidden predictive information from large databases. One of the central tasks associated with data mining is to discover profitable actions from the dataset for the decision maker. Discovering these actions can be accomplished through extracting Action Rules from the data, which has become an attractive research topic in data mining. Several methods have been developed for the discovery of Action Rules and variety of methods for association rules in the past few years. However, with the explosive recent growth of the amounts of data, there is a need for the development of scalable methods for Action Rules discovery to accommodate the massive datasets. We are not aware of any such methods existing at this time. In this paper, we propose a novel approach - a parallel Action Rule discovery algorithm based on MapReduce paradigm through count distribution. We use Hadoop, as a scalable and distributed framework for implementing this method. Experiment shows much faster computational time for Action Rules discovery in a distributed environment compared to the traditional single machine method.

Keywords—Action Rules; MapReduce; Apriori; Count Distribution

I. INTRODUCTION

Discovering useful and actionable knowledge from data is gaining a lot of interest in recent years. Action Rules are a special type of rules, which are suitable for suggesting actions the user can undertake to his/her advantage. An Action Rule is a rule extracted from a decision system that describes a possible transition of objects from one state to another with respect to a distinguished attribute called a decision attribute [1]. There are two types of attributes in a decision system, namely: flexible and stable. The flexible attributes, as the name implies, can be changed. This change can be influenced and controlled by users. The main application is: to provide an insight of how values of some attributes should to be changed so the undesirable objects can be shifted to a desirable group or class. For example, making profit from a business could be considered as an application of Action Rules. The Action Rules extracted from business data suggest us what changes within the flexible attributes are needed to achieve the desired goal.

Discovering Action Rules has become an attractive research topic in data mining. Several methods have been developed for the discovery of Action Rules in the past few years. However, with the explosive recent growth of amounts of data, there is a need for the development of scalable methods for Action Rules discovery to accommodate the massive datasets. With emerging trends in cloud computing, massive data storage, data mining is a research field with a very practical value. In this work we propose a novel approach for adapting the Apriori Action Rules mining algorithm to processing in a distributed environment where the Action Rules generation is done in parallel on a number of clustered nodes in a distributed processing environment. For the purpose of processing the Action Rules mining in a distributed environment we are utilizing the MapReduce [2] framework on an Apache Hadoop platform, which allows for rapid processing of large amounts of data on clustered computing nodes. The proposed method provides for high scalability and efficiency for the traditional Action Rules discovery data mining algorithm. The rest of the paper is organized as follows: in section II, we review the related literature, in section III, we describe the methodology of our proposed distributed MR-Apriori Action Rules algorithm, in section IV, we show the results of the implementation of the proposed method, and finally in section V, we conclude and discuss directions for the future.

II. RELATED WORK

Authors Z.W. Ras and A. Wieczorkowska introduce the concept of Action Rules in [1]. Action rules are further investigated in [3], [4], [5], [6], [7], [8] and [9]. In most of these works, Action Rules are constructed from pre-existing classification rules. Algorithms like LERS [10] or ERID [11], are used to initially generate classification rules. Later Action Rules are constructed by combining one or more of these classification rules. Paper [12] is probably the first attempt towards formally introducing the problem of mining Action Rules without pre-existing classification rules. Their proposed algorithm is similar to Apriori algorithm. The algorithm is formulated as a search problem in a support-confidence-cost framework in paper [12]. Algorithm ARED, presented in [13], is based on Pawlak's [14] model of an information system S. Its goal is to identify certain relationships between granules defined by the indiscernibility relation on its objects. Some of these relationships uniquely define Action Rules for the information system S. Papers [15], [13] present a new strategy for discovering Action Rules directly from the decision system. In [15], Action
Rules are built from atomic expressions following a strategy similar to ERID [11]. In paper [16], authors present a strategy for extracting action rules directly from a decision system, called Association Action Rules (Apriori).

Most of these algorithms are not designed to work with very large amounts of data. Hence with today’s fast growing massive datasets, the above mentioned algorithms for Action Rules discovery take considerable amount of computational time. In this paper, we propose a novel approach for adapting the Association Action Rules mining algorithm to processing in a distributed environment. We use MapReduce framework for providing scalability to the Association Action Rules mining algorithm to very large datasets. MapReduce [2] is a programming model introduced by Google in 2004 to support distributed computing on large data sets on clusters of computers. Authors in paper [17] propose a distributed Association Rules algorithm based on MapReduce programming model (MR-Apriori). They show that MR-Apriori algorithm performs effectively on Hadoop distributed computing environment. In this paper, we implement MR-Apriori Count Distribution algorithm for action rule discovery.

III. METHODOLOGY

Before getting into the details of the MR-Apriori algorithm, we give an overview of the Association Action Rules which uses the Apriori strategy in building action rules.

A. Association Action Rules (Apriori)

Association Action Rule algorithm (Apriori) is based on extracting action rules without any pre-existing classification rules [16]. The concepts of Action Sets and frequent Action Sets are used in this algorithm. These are best explained by an example information system as shown in Table 1. Let's call the information system in the Table 1. - S. From the table, we can see that S has a set of objects X = {x₁, x₂, x₃, x₄, x₅, x₆, x₇, x₈} set of attributes A = {a, b, c, d} and set of values V = {a₁, a₂, b₁, b₂, c₁, c₂, d₁, d₂}. Consider attribute [b] to be flexible, attributes [a, c] to be stable, and attribute [d] to be the decision attribute in S. Values of flexible attributes can be changed. Values of stable attributes cannot be changed. Now, let's see how we build Action Rules from Action Sets. First, we introduce the concept of an Atomic Action Set. The expression (a, a₁ → a₂) is an example of an Atomic Action Set. It means that: the value of attribute a changes from a₁ to a₂. Collection of these Atomic Action Sets are Action Sets. For example, (a, a₁ → a₂),(b, b₁ → b₂),(d, d₁ → d₂) is an Action Set which includes a decision attribute d. We can compose Action Rules from these Action Sets which include the decision attribute, decision_from and decision_to values. An Action Rule composed from the above action set is (a, a₁ → a₂),(b, b₁ → b₂) → (d, d₁ → d₂). For constructing all Action Rules from a particular dataset, we need to find all possible combinations of Action Sets which involve the decision attributes and required values (decision_from and decision_to) of the decision attribute. Apriori algorithm is the most widely used algorithm for identifying association rules. It uses an iterative method to generate (k+1) item-sets from k item-sets. Constructing Action Sets is very similar to generating item-sets in association rules. Hence we use an Apriori based strategy for building Action Rules in this method.

It is necessary to understand some commonly used interpretations of Action Sets before we explain how to construct them. Paper [16] explains these notations and talks about how support and confidence of action sets are calculated.

Standard interpretation N₅ of action sets in S is defined as follows:

1. If (a, a₁ → a₂) is an Atomic Action Set, then N₅ ((a, a₁ → a₂)) = [{x ∈ X: a(x) = a₁}, {x ∈ X: a(x) = a₂}].
2. Let t₁ = (a, a₁ → a₂) · t and N₅ (t₁) = [Y₁, Y₂], then N₅ (t₁) = [Y₁ ∩ {x ∈ X: a(x) = a₁}, Y₂ ∩ {x ∈ X: a(x) = a₂}].

Let us define [Y₁, Y₂] ∩ [Z₁, Z₂] as [Y₁ ∩ Z₁, Y₂ ∩ Z₂] and assume that N₅ (t₁) = [Y₁, Y₂] and N₅ (t₂) = [Z₁, Z₂]. Then, N₅ (t₁ · t₂) = N₅ (t₁) ∩ N₅ (t₂).

If t is an Action Set and N₅ (t) = [Y₁, Y₂], then the support of t in S is defined as:

$$\text{Sup}(t) = \min \{\text{card}(Y₁), \text{card}(Y₂)\}.$$  

Now, let r = [t₁ → t₂] be an action rule, where NS (t₁) = [Y₁, Y₂], NS (t₂) = [Z₁, Z₂]. Support and confidence of r are defined as follows:

$$\text{Sup}(r) = \min \{\text{card}(Y₁ ∩ Z₁), \text{card}(Y₂ ∩ Z₂)\}$$

$$\text{Conf}(r) = \frac{\text{card}(Y₁ ∩ Z₁) \cdot \text{card}(Y₂ ∩ Z₂)}{\text{card}(Y₁) \cdot \text{card}(Y₂)}$$

![Figure 1. Support and Confidence of Action Rules.](image)

We build Association Action Rules by constructing all possible Action Sets which involve the decision attribute and required decision values. For constructing all possible combination of Action Sets, we initially start with Atomic Action Sets and find their support.
and confidence using the steps mentioned above. Some example Atomic Action Sets from Table I. with their support values are:

\[
\begin{align*}
(a, a_1) & \quad \text{support 2} \\
(a, a_2) & \quad \text{support 6} \\
(b, b_1) & \quad \text{support 4} \\
(b, b_2) & \quad \text{support 4} \\
(b, b_1 \rightarrow b_2) & \quad \text{support 4} \\
(b, b_2 \rightarrow b_1) & \quad \text{support 4}
\end{align*}
\]

In a similar fashion, the rest of the Atomic Action Sets are constructed. Note that, a is stable and b is flexible attribute. Once all Atomic Action Sets are generated, we combine possible combinations of Action Sets to generate larger Action Sets and look for those sets which contains the required decision change ((decision_from and decision_to) of the decision attribute).

In the example from Table I., d is the decision attribute. If Action Rule specifies a decision change from d1 to d2, we look for all Action Sets which contains (d, d1 \rightarrow d2) and compute the support and confidence of those action sets. If the support and confidence are greater than the user specified threshold, then Action Sets are considered frequent, and we retain them. Otherwise, we remove them. An example Action Set from Table I. is shown below:

\[
(a, a_2) \cdot (b, b_1 \rightarrow b_2) \cdot (c, c_1) \cdot (d, d_1 \rightarrow d_2) \quad \text{support 2}
\]

Finding all such possible Action Sets iteratively requires a lot of memory and time, and hence we propose the use of distributed computing environment, in order to provide scalability of the Apriori Action Rules algorithm to today’s increasingly large datasets. We are not aware of any other existing methods for computing Action Rules in distributed environment.

**B. MR-Apriori Count Distribution Algorithm for Parallel Action Rules Discovery**

In this method, Action Rules based on Apriori are transformed to work in a distributed setup using MapReduce. Apriori algorithm finds all possible Action Set combinations by scanning the database time after time, which consumes a lot of time and memory space for massive data. Hence researchers have been exploring ways to apply the MapReduce model to this field [17, 18]. With Count Distribution stated as the best way to parallelize the Apriori algorithm, we propose an implementation of MR-Apriori Count Distribution Algorithm for Parallel Action Rules discovery in this paper.

We use Hadoop framework to run the Apriori algorithm parallel in distributed cluster. Hadoop is a free, Java-based programming framework that supports the processing of large data sets in a distributed computing environment. The core of Apache Hadoop consists of a storage part, known as Hadoop Distributed File System (HDFS), and a processing part called MapReduce. Hadoop splits files into large blocks and distributes them across nodes in a cluster. To process data, Hadoop transfers packaged code for nodes to process in parallel based on the data that needs to be processed.

Count Distribution algorithm uses a simple principle of allowing redundant computations in parallel [18]. Computations here involves the construction of action rules and finding their support and confidence. We use the same techniques described in Association Action Rules section for building action rules with their support and confidence. But in MR-Apriori Count Distribution method, data is distributed as blocks across different nodes. Same functions for building action rules, computing support and confidence are executed parallel across nodes for all blocks of locally distributed data. These are included in the Map task in MR-Apriori algorithm.

Output from all map nodes are synchronized to construct the final action rules with global support and confidence, which is included in the Reducer class. Fig 2 gives a high level overview of the implementation of MR-Apriori Count Distribution Algorithm for parallel action rules discovery.

Before moving to the implementation of MR-Apriori, we briefly review the MapReduce programming paradigm and the methods in Mapper and Reducer classes. MapReduce is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster. A MapReduce program is composed of a Map task that performs filtering, sorting, etc. and a Reduce task that performs a summary operation as shown on Figure 2. Map task is coded in Mapper class which has methods like setup, map, cleanup and run. For the map task, framework calls setup once, then map for each record and finally cleanup once at the end of task. Similarly, Reduce task is written in Reducer class which has methods like setup, reduce, cleanup and run and executes similar to map. Both map and reduce operate over key-value pairs.

The proposed MR-Apriori Count Distribution Algorithm for Parallel Action Rules Discovery is shown on Figure 3. and involves the following steps:

a) First the dataset for which action rules are to be discovered is stored in HDFS. A text file which contains the name of attributes of dataset and another text file which contains user parameters like stable attributes, decision attribute, decision_from, decision_to, minimum support and confidence, are stored in the Distributed cache. These two text files are read by every node before the execution of map functions. We name the first file as ‘attributes.txt’ and the other text file as ‘parameters.txt’.
b) The dataset is divided horizontally into m blocks and are distributed to m nodes for processing. The default block size is 64MB. However, this size can be configured manually as well.

c) In the MapReduce phase, initially Map tasks executes for all input splits across different nodes in the cluster. For the Map task, setup(), map() and cleanup() methods are overridden. First setup() is called once during execution. It reads the text files from Distributed Cache and stores the attribute names and user parameters. Then map() function is called for each record. Map() reads each record as key-value pair and stores each record in a data structure which is defined globally in Mapper class. Data structure we use is a hashmap. For the first record in Table I, map() function produces output in the following format:

\[
\{ \{ a_{a1}, b_{b1}, c_{c1}, d_{d1} \} , \text{count} \}
\]

The output of map() is a key-value pair and count variable just counts the number of occurrences of same rows. Once map() function reads all the records in that split and stores these records in a hashmap, cleanup() function is called. Cleanup() contains the methods to build action rules from the data stored in hashmap using Apriori strategy and compute their supports and confidence as described in Association Action Rule section. Output from cleanup() is the output of the map task, which are the locally found Action Rules for the input splits. An example output of data from Table I is shown below.

\[
\{ \{ (a, a2) \rightarrow (b, b1) \rightarrow (c, c1) \rightarrow (d, d1 \rightarrow d2) \} ,
\{ 2, 44.44 \}
\]

d) After the completion of map tasks for all splits across all nodes, output from these nodes is sorted and shuffled. This is the shuffle phase and is taken care of by the framework itself. Output from the shuffle phase if given as the input to the reduce phase.

e) Reduce phase gets the input as [Action Rules, List (support_confidence)] pairs. In reduce() these records are read initially and the final support and confidence are aggregated for every unique action rules. Supports for same Action Rules are added to get the final support in reduce() function. Output from reduce() is the reduce task output and an example format is shown below:

\[
\{ \{ (a, a2 \rightarrow a1) \rightarrow (b, b1 \rightarrow b2) \rightarrow (d, d1 \rightarrow d2) \} ,
\{ \text{Support} = 6, \text{Confidence} = 84.4\% \}
\]

IV. EXPERIMENT

For testing the performance of the proposed method, we execute the algorithm on the UNCC-Charlotte Research Hadoop cluster. We use the Car Evaluation dataset from UCI Machine Learning Repository [19]. Dataset originally consists of 1728 records. For our experiment, we replicate the records in the dataset 125 times to test this algorithm with 216,000 records. Dataset has a total of 7 attributes which are: car buying price (buying), maintenance price (maint), number of doors (doors), persons capacity (persons), size of luggage boot (lug_boot), safety of the car (safety) and car acceptability (class). Car acceptability is the decision attribute which has values as ‘unancc’, ‘acc’, ‘good’, ‘vgood’. Missing values are removed before we begin processing. For the experiment, we use a Hadoop cluster which consists of 73 nodes.

Time taken to the complete execution using this cluster compared to a traditional method (using a single machine) is recorded in Table II. We can see that the proposed

<table>
<thead>
<tr>
<th>Method</th>
<th>Number of nodes</th>
<th>Time taken(in min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional Apriori</td>
<td>1</td>
<td>9 min 23 sec</td>
</tr>
<tr>
<td>MR-Apriori – Hadoop</td>
<td>73</td>
<td>1 min 26 sec</td>
</tr>
</tbody>
</table>

TABLE II. COMPARISON OF COMPUTATIONAL TIME FOR APRIORI ACTION RULES DISCOVERY
MR-Apriori Count Distribution Algorithm for Parallel Action Rules Discovery, which runs in a distributed environment, significantly improves computational time compared to the traditional Apriori Action Rules extraction algorithm, running on a single machine. Therefore, the proposed method increases the scalability of the traditional Apriori Action Rules discovery algorithm.

The Action Rules discovered in this experiment have the decision attribute set to: acceptability (class attribute), where we would like to change value of the decision attribute from ‘unacc’ (unacceptable) to ‘acc’ (acceptable). This means the discovered Action Rules provide suggestions to the user, in his case a car manufacturer, of how the car can be improved from unacceptable to acceptable condition.

Example output Action Rule we obtained after the execution are shown in Figure 4, with their respective support and confidence values. If we look at the first Action Rule shown in Figure 4: (doors, 2 → 4) → (safety, low → high) → (persons, 2 → 4) → (class, unacc → acc) [Support: 1508.0 ; Confidence: 64.17%] in this means that if the number of door are changed from 2 to 4 and the safety is increased from low to high, and the number of persons the car fits is increased from 2 to 4, then the decision (class) attribute is expected to change from unacceptable to acceptable state. Therefore, providing suggestion to the car manufacturer of how the vehicle can be improved. 1508 tuples support this Action Rule, and the confidence is 64.17%.

V. CONCLUSION

In this work, we propose a novel method, which allows for parallel discovery of Action Rules in a distributed environment utilizing MapReduce paradigm. We use Apriori strategy for Action Rules extraction and Count Distribution Algorithm for implementing the parallel processing. This is an original and novel approach which allows for distributed extraction of Action Rules, and greatly improves the scalability of Action Rules extraction with large datasets. We are not aware of any other existing algorithms for distributed Action Rules extraction. The results shows that the proposed method performs effectively on Hadoop clustered computing environment, and that the computational time for Action Rules discovery in distributed environment is substantially faster than the traditional Action Rules discovery using a single machine. Thus providing scalability for Action Rules discovery to accommodate today’s increasingly large datasets.

In addition to the industrial dataset, which we used (the CarEvaluation dataset), the proposed algorithm can be applied to financial data – for example to suggest how customer loyalty can be increased, or how loan risk can be decreased; it can be applied to medical data – for example to suggest how a particular disease can be cured (e.g. heart disease, or breast cancer); it can be applied to social networks data – for example to provide suggestions of how to increase the interest of one friends community in another friends community.

Future work includes testing the proposed algorithm with the above mentioned types of datasets in financial, medical, and social network domain. Future work also includes extending this study to adapt the Apriori Action Rules algorithm for parallel processing in Spark distributed framework, which allows for keeping large working datasets in memory between jobs, and may therefore outperform the equivalent MapReduce workflow.

REFERENCES


