Design of Knowledge Acquisition Subsystem with Mining Association Rules from Big Data

Giang-Truong Nguyen, Van-Quyet Nguyen, Kyungbaek Kim

Department of Electronics and Computer Engineering
Chonnam National University
Gwangju, South Korea
truongnguyengiang.bk@gmail.com, quyetict@utehy.edu.vn, kyungbaekkim@jnu.ac.kr

Abstract

Expert system has played an important role in human's life for a long time. In order to establish an expert system, knowledge acquisition is the process which should be considered first. Regarding to this step, two of the most demanding requests are: how to discover the knowledge from a massive dataset, and how to organize them in a temporary database properly in order to be reviewed again. In this paper, we propose a design of a subsystem, which acquires the knowledge from Big Data sources and stores them following the rule form. Our work is conducted with the support of a data mining technique called association rules mining, which is executed through parallel distributed environments: given data under file form as input, outputs which includes rules representing knowledge with their corresponding certainty factor are gotten. Then all of these rules are stored in a relational database, which could be queried out by knowledge engineers to review them again before encoding them into the knowledge base. By the experiments with real dataset, it is confirmed that our proposed design could not only acquire the knowledge from Big Data source in an acceptable duration, but also handle well with the querying out situation when a lot of rules are gotten.

Keywords: Big Data, Expert System, Association rules mining

I. Introduction

Expert systems [1] have attracted much attention in recent decades. In the processes for establishing these systems, knowledge acquisition subsystem plays a very important role: it has the responsibility for acquiring knowledge from many sources, then structuring and letting them to be rechecked before encoding them into the knowledge base. The most efficient source of data for acquiring the knowledge is from human expert [7], which is supposed to cost not only much time, but also money. Consequently, expert system maker would want some alternative solutions which could acquire the knowledge automatically without much cost like the human expert source.

There have been a lot of researches regarding automating orientation. Herskovits et al [8] proposed calculating entropy value from the inputted database of cases to produce a belief network. Leung et al [9] employed association rules mining in incomplete information system. Those methods were proved to be efficient years ago, but they could not be satisfying when dealing with big volume data issue at this time.

Association rules mining has been used for acquiring knowledge for a long time. The original work of this technique was from the shopping baskets of the retainers: giving a collection of transactions with their corresponding purchased items, mining association rules is finding which items customers could pick; which are considered as consequent; after grabbing some ones known as antecedent. This mining could only be conducted after the prior work is done: finding the frequent itemsets; which are the set of items accompanying together whose rate of appearances over the overall number of transactions (a.k.a support) is larger than a given threshold (which is also known as minimum support).

In this paper, we propose a design of knowledge acquisition subsystem for expert system by using association rules mining, which could be applied specifically on big dataset. Our work is executed in 3 main processes. The first one is *Raw Big Data Processing*, which will handle the raw data to get the so-called transactions with their corresponding items. Afterward, the association rules mining process will initially, with the input minimum support, find all the frequent itemsets, which is the basement for finding association rules. Finally, those results will be stored by the rules Database, with a proper design satisfying the possible large quantity exploding in the future. By this proposed database, users can query and verify rules' validity before applying them in their expert system.

II. Background

A. Expert System

Expert System is a kind of system which is used to support making up decisions on some specific domains based on human knowledge, which is discovered in the real life from human experts or real data exploration. Fig. 1 describes an Expert System architecture, which includes some components. They could be more or fewer in other variants, but Fig. 1 is one of the most typical one. In this architecture, the Knowledge Acquisition subsystem gets the responsibility to accumulate knowledge from Real Life Data or Human Experts. Afterwards, knowledge is verified before being encoded into the Knowledge Base. Users will interact with the Expert System by sending requests to the component called User Interface. From here, based on the knowledge accumulated in the Knowledge Base component, the Inference component will try to find the best suggestion for the requests from users and the Explanation Component will explain why that suggestion is

chosen.

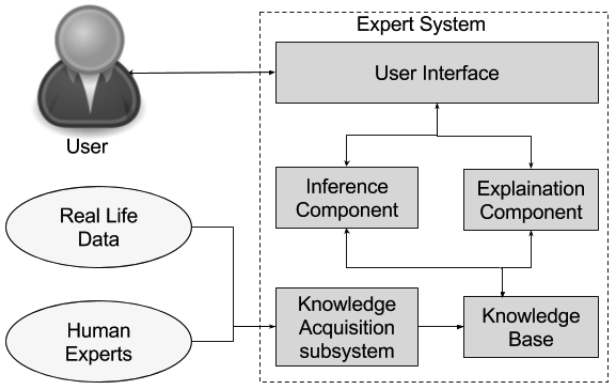

Fig. 1 Architecture of a typical Expert System

B. Association rules mining

In this technique, two entities should be considered: a set of items $I = \{I_1, I_2 \ldots, I_n\}$ and a collection of transaction $T = \{T_1, T_2 \ldots, T_n\}$ with each of them contains (some) independent item(s). Mining the association rules includes 2 main tasks: frequent itemsets finding and association rules finding. The aim of the former is trying to get all the set of items whose rate of appearance (*support*) is larger than a given threshold, which is called *minimum support* (*minsup*). In order to do this work, one of the most well-known methods is Apriori algorithm [2], which employs a pruning technique to solve the problem. In any levels of itemset length, it will find all the possible sets which satisfy the minimum support requirements. Subsequently, in the next levels, the itemsets, whose length is 1-longer than the previous level's length, will have to contain all subsets being the found frequent itemsets previously. Apriori algorithm is very useful in the old days when parallel computation [3] was not popular. However, it is not suitable anymore those days because Apriori can only be conducted on a single machine environment only.

SON algorithm [4] described in *Algorithm 1* is supposed to work well on parallel environment because in the first pass it divides the dataset into smaller non-overlapping fragments, then sequentially on each of them finds the *local frequent itemsets*, which will be aggregated together to become the *global frequent itemset candidates*. In the next pass, these found candidates are considered on the whole dataset to find their overall appearances, which are used to decide if these candidates are really frequent or not. Our work applies the SON algorithm not on a single machine, but in a parallel environment, specifically Hadoop [5].

After that, the association rules finding will continue the work based on what has been found. Consider a frequent itemset *l* and one of its subset *a*, the so-called *confidence* value of any rule is calculated by:

$$confidence(a \rightarrow (l-a)) = \frac{support\ (l)}{support\ (a)}$$

If the confidence value of any rule is larger than a given threshold, this rule is considered as association rule.

Algorithm 1: SON algorithm

***Input**: Set of items $I = \{I_1, I_2, \ldots, I_n\}$;*

collection of transaction $T = \{t_i , t_i \ c \ I\}$;

minimum support threshold minsup.

***Output**: itemsets whose support is larger than minsup*

1. Dividing *T* into m parts
2. **Initialize** *local_sets* ← null; *global_sets* ← null
   // local_sets: global frequent itemset candidates
   // first passing
3. **for** *i* ← 1 **to** *m*
4. *temp_local_sets*←get_frequent_itemsets(*i*)
5. *local_sets*.add(*temp_local_sets*)

   //second passing
6. **for** *i* ← 1 **to** *m*
7. *part* = get_part(*T* , *i*)
8. **for** *j* ← 1 **to** *local_sets*.length()
9. *local_sets[j]*.quantity ← *local_sets[j]*.quantity + get_quantity(*part* , *local_sets[j]*)

10. **for** *i* ← 1 **to** *local_sets*.length()
11. *current_set* ← *local_sets[i]*
12. *support* ← *current_set*.quantity() / *T*.length()
13. **if** (*support* > *minsup*)
14. *global_sets*.add(*current_set*)
15. **return** *global_sets*

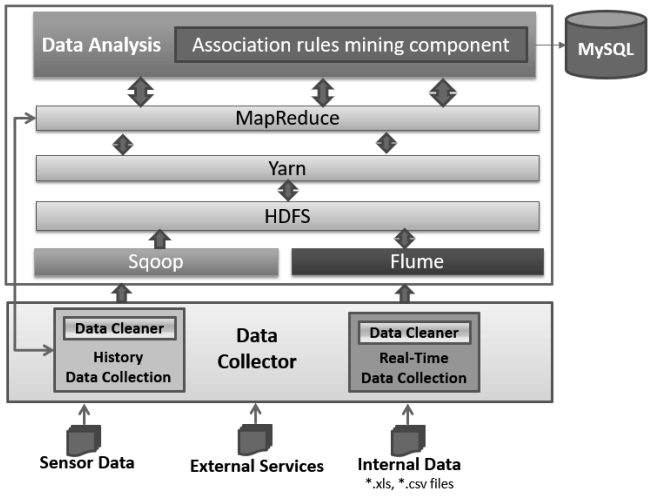

Fig. 2 Big Data Platform for pre-processing and mining the rules

III. Design of Knowledge Acquisition subsystem

A. Overview of our knowledge acquisition subsystem on Big Data platform

Our knowledge acquisition system uses 3 main processes for acquiring knowledge. Initially, the data from file source is pre-processed to get the transaction form with their items accompanying together. Afterwards, those transactions data will be handled to find the frequent itemsets first, which will be the basement for finding the association rules. Finally, these rules will be stored in the relational database.

In order to conduct these processes, we employ a Big Data platform which has been introduced in our research before [6]. Fig. 2 describes our design of knowledge acquisition

subsystem on this platform, which is created on the basis of Hadoop and modified from [6] to satisfy the aim of finding knowledge.

This subsystem will get the raw data from External Services, Sensors as well as Internal Data like csv, xls files. Those data, after being some basic processes, will be stored in Hadoop distributed file system (HDFS) with the support of a component called MapReduce, which divides the whole dataset into smaller parts, then execute on each of them. Subsequently, pre-processing phase is conducted by MapReduce component, which stores the transactions data in HDFS. Then, the association rules mining phase will be executed by the association rules mining component inside Data Analysis, which reads the data from this storage and runs the SON algorithm in the parallel environment to get the results. Finally, they will be stored in MySQL database.

B. Raw Big Data Pre-processing

The aim of this component is getting the data, which is the form of transactions with corresponding items accompanying together, to mine the association rules. In the scope of this paper, the input data is limited to file form, which contains data about some domains whose knowledge is needed to be extracted. Each input file has many rows, which stand for the accumulated data. By manually analyzing the input data, (some) field(s) can be possibly found, which could establish unique objects with unique identities representing for unique transactions. Afterwards, among other fields, one can be picked to make the items data for each transaction. These fields can be considered as transaction fields and item field respectively. In section V, we include an example with real dataset for getting the transaction data.

With the support of MapReduce, this work can be done parallelly. The raw big data, after being stored in HDFS by MapReduce, can be split into some smaller parts. Based on the knowledge of the executor, the fields chosen for making transaction identity and item name will be set. The first map procedure will produce the pair <key, value> whose key is gotten from transaction fields, and value is gotten from item field. After the reduce procedure, all the item will be grouped together with the same key, making the list of transactions with corresponding items. The final result of this component is stored in HDFS, which allows the next component to read and process with MapReduce procedure.

C. Association rules mining

As mentioned before, our method is conducted based on SON algorithm on the parallel computing environment with Hadoop. In order to conduct this work, there should be 2 phases of MapReduce to handle 2 passes of the SON algorithm.

In the first phase, the overall gotten list of transactions is divided into many smaller non-overlapping parts, which are handled by some mappers to find the *local frequent itemsets* on each smaller part. This procedure's output has the form <key, value> whose key is the item's name and value is 0 because this phase is just used for finding the possible frequent candidate, so its number of appearances need not to be considered. After being processed by the reducer, the overall *global frequent itemset candidates* are gotten.

Afterward, in the second MapReduce procedure, the list of *global frequent itemset candidates* is employed, which is assigned for each mapper. After the map procedure, the appearances of each itemset of *global frequent itemset candidates* on each smaller part will be gotten first, then after the reduce phase, their overall appearances over the whole dataset will be achieved. Based on this overall achievement, the support of each *global frequent itemset candidates* can be calculated, which decides which itemsets are frequent or not.

Finally, in order to accomplish the task of mining association rules, based on the found frequent itemsets, the rules will be gotten by calculating their confidence. In this step, we follow the work mentioned in [4]: generating subset of each frequent itemsets by each level of subset length decreasing. If any rules containing (a) as antecedent and (l - a) as consequent could not satisfy the minimum confidence requirement, all the rules containing *a's* subset a_{subset} as antecedent and (l - a_{subset}) as consequent will be discarded without considering.

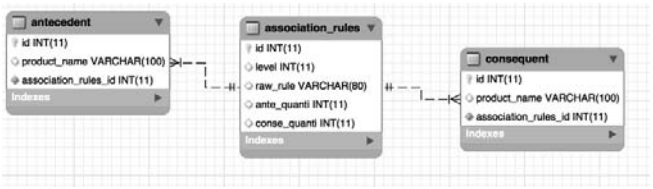

Fig. 3 Design of MySQL Database for storing association rules

D. Storing the archived rules in MySQL database

After getting all the association rules from the dataset, these rules will be stored in the database. Specifically, we employ a relational database known as MySQL. The design of the database for storing achieved rules is described in Fig. 3.

As seen from Fig. 3, there are 3 tables to store the rules' information. The first one is *association_rules* which stores raw rule's content *raw_rule* (the string format of the rules: antecedent ⇒ consequent), number of items in the consequent *conse_quanti*, number of items in the antecedent *ante_quanti* and rule's confidence level *level*, which means the confidence of the found rule. The other tables are *antecedent* and *consequent*, which stores each individual item of antecedent and consequent of the rules. In each rule, there could be some items in the antecedent and some items in the consequent.

Because those rules will be later queried again to be reviewed and inserted into the knowledge base, we create a design which supports mainly for querying based on the chosen items in the antecedent as well as consequent. With the support of indexing function from MySQL, it would take users less time for querying the wanted results, even when the number of gotten rules explodes later.

V. Evaluation

We conducted our evaluation with a Hadoop cluster on 5 machines: one for master node and 4 for computing nodes. Each machine has 4 CPU and 16 GB of RAM. The data set used for testing is taken from agriculture data in Korea in 2015. In this dataset, there are many rows, each of which have farms information like their identity number, their grown products, their grown area and income. Because a specific farm with a unique identity number could have many grown products, there could be many rows with the same identity number, but different grown product. Consequently, farms' identity number and grown products are chosen as transaction field and

item field respectively. Moreover, in order to increase the quality of the gotten rules, we only get the transactions from farm whose income is larger than 10,000 thousand won. This filter proves that the gotten rules are retrieved from good sources.

Table 1: Evaluating for the pre-processing

Number of records	Number of gotten transactions (with income condition)	Number of gotten transactions (without income condition)	Executed time (s) (with condition / without condition)
4,053,802	281,055	419,569	31/24
6,082,807	390,125	642,614	33/28
8,110,107	489,678	909,954	35/32
10,100,124	566,257	1,261,391	38/36
11,275,355	614,736	1,561,632	42/39

The first evaluation is conducted with pre-processing data. The results for this evaluation is shown in Table 1. As seen from this table, the executed time also increases when the number of records increases. However, the increasing of executed time is marginal: from 24 to 39 seconds without the income condition, and from 31 to 42 seconds with the income condition, while inputted records quantity is nearly doubled.

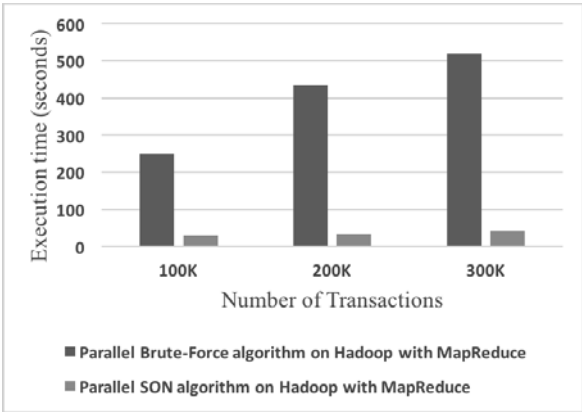

Fig. 4 Comparison between Brute algorithm and SON algorithm conducted on Hadoop environment

Subsequently, another evaluation is made with the frequent itemset finding phase, by comparing with an original method called Brute-Force. In this one, each possible itemset is listed, then their appearances are counted by looking at each transaction. Both of our method and the original method are conducted on proposed Hadoop environment. The result is shown in Fig. 4. From this figure, it is observable that the execution time of our method is always much lower than the original method when the number of transactions increases. The minimum support used for this case is 0.1.

Finally, the last evaluation is conducted with the queried time when a search function is made. It is assumed that a user wants to search some rules, and he will input some items, so the result he wants are some rules which includes his chosen items as antecedent. This evaluation is conducted with the number of items in the antecedent increases, and nearly 300,000 gotten rules. As seen from Fig. 5, which is the result of this evaluation, when the number of chosen items in the antecedent increases, the queried time increase mildly, which proves that our design of the database for storing the gotten rules works properly.

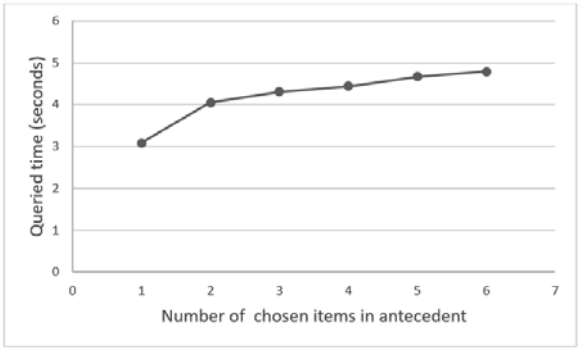

Fig. 5 Queried time of rules searching

VI. Conclusion

In this paper, we have presented a design for the knowledge acquisition subsystem, which is used for accumulating rules for expert system. The raw big data is pre-processed to get the data under transactions form with corresponding items; then from this gotten data, association rules are gotten, which later are stored in relation database with a proper design. Our future work will focus on improving the performance for finding the rules more quickly.

Acknowledgment

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT & Future Planning(NRF-2017R1A2B4012559). This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2017-2016-0-00314)

References

[1] Jackson, Peter. "Introduction to expert systems." (1986).
[2] MLA Agrawal, Rakesh, and Ramakrishnan Srikant. "Fast algorithms for mining association rules." Proc. 20th int. conf. very large data bases, VLDB. Vol. 1215. 1994.
[3] en.wikipedia.org/wiki/Parallel_computing
[4] Savasere, Ashok, Edward Robert Omiecinski, and Shamkant B. Navathe. An efficient algorithm for mining association rules in large databases. Georgia Institute of Technology, 1995.
[5] hadoop.apache.org/
[6] Van-Quyet, Nguyen, et al. "Design of a Platform for Collecting and Analyzing Agricultural Big Data." Journal of Digital Contents Society 18.1 (2017): 149-158. APA
[7] engineering.purdue.edu/~engelb/abe565/knowacq.htm
[8] Herskovits, Edward H., and Gregory F. Cooper. "Kutato: An entropy-driven system for construction of probabilistic expert systems from databases." arXiv preprint arXiv:1304.1088 (2013).
[9] Leung, Yee, Wei-Zhi Wu, and Wen-Xiu Zhang. "Knowledge acquisition in incomplete information systems: a rough set approach." European Journal of Operational Research 168.1 (2006): 164-180. APA