





Article

Incremental Algorithm for Association Rule Mining under Dynamic Threshold

Iyad Aqra ^{1,*}, Norjihan Abdul Ghani ^{1,*}, Carsten Maple ^{2,3,†}, José Machado ^{4,‡} and Nader Sohrabi Safa ^{5,‡}

¹ Department of Information Systems, Faculty of Computer Science & Information Technology, University of Malaya, Kuala Lumpur 50603, Malaysia

² Cyber Security Centre at WMG, University of Warwick, Coventry EC4A 3BZ, UK; cmaple@turing.ac.uk

³ The Alan Turing Institute, The British Library, London NW1 2DB, UK

⁴ ALGORITMI Research Center, University of Minho, 4710-057 Braga, Portugal; jmac@di.uminho.pt

⁵ Faculty of Engineering, School of Computing, Electronics and Mathematics, Coventry University, Coventry CV1 5FB, UK; sohrabisafa@yahoo.com

* Correspondence: i_aqra@siswa.um.edu.my (I.A.); Norjihan@um.edu.my (N.A.G.)

† These authors contributed equally to this work.

‡ These authors contributed equally to this work.

Received: 19 November 2019; Accepted: 4 December 2019; Published: 10 December 2019



Abstract: Data mining is essentially applied to discover new knowledge from a database through an iterative process. The mining process may be time consuming for massive datasets. A widely used method related to knowledge discovery domain refers to association rule mining (ARM) approach, despite its shortcomings in mining large databases. As such, several approaches have been prescribed to unravel knowledge. Most of the proposed algorithms addressed data incremental issues, especially when a hefty amount of data are added to the database after the latest mining process. Three basic manipulation operations performed in a database include add, delete, and update. Any method devised in light of data incremental issues is bound to embed these three operations. The changing threshold is a long-standing problem within the data mining field. Since decision making refers to an active process, the threshold is indeed changeable. Accordingly, the present study proposes an algorithm that resolves the issue of rescanning a database that had been mined previously and allows retrieval of knowledge that satisfies several thresholds without the need to learn the process from scratch. The proposed approach displayed high accuracy in experimentation, as well as reduction in processing time by almost two-thirds of the original mining execution time.

Keywords: data mining; knowledge extraction; association rule mining; incremental mining; dynamic threshold

1. Introduction

This study addresses association rule mining (ARM), which refers to a widely known data mining technique. Data mining extracts useful patterns that are hidden in a huge database. The reckoned forms for the knowledge are a frequent pattern (frequent itemset) and a rule set. Nevertheless, acquiring hidden knowledge from a massive database is a complicated process [1]. Some of the proposed data mining strategies and techniques are ARM [2–8], classification rules [9,10], clustering rules [8], and sequential patterns [11,12]. The ARM, in particular, unravels correlations between various attributes (items) in transaction databases [2,5,6,8,13]. Applications of ARM include the market basket and a great deal of issues that demand discovery of hidden knowledge, such as protein sequences [14], medical diagnosis [14], energy [3,15,16], smart homes [11], and other vast industrial fields [17]. The Apriori algorithm has been acknowledged as an icon implementation for the ARM

technique [11]. The Apriori algorithm generates a frequent itemset that is determined by testing the candidate itemsets. When an itemset within the candidate itemset has weight exceeding the minimum support threshold, it is considered as a frequent itemset. After generating all frequent itemsets, the association rules are retrieved from the gathered frequent itemsets. Since rules have confidence and a user defines the minimum confidence threshold, any rule with confidence exceeding its minimum level is considered as a strong rule set. Data mining is a highly attractive field and has become an even more popular topic amongst researchers since the introduction of ARM in the work carried out by Agrawal (see [2,13]). The data mining processes have been continually improved and developed by a substantial number of studies. In fact, a significant correlation has been established between ARM and other fields [6,8], namely machine learning [18], association rule discovery, clustering rules [19,20], classification rules [6,19,21], associative techniques [19], sequential patterns [21], and decision trees [22]. Amidst these, looking for common itemsets within a massive database has become a rather common method in data mining, where the preponderance of data mining techniques is associated with discovering frequent patterns, which serve as the main output for ARM. These techniques arrange the frequent itemsets and rules in a timely manner. Data mining has been implemented in the high business domain to increase its value, such as Internet of things and sensors [3]. One popular area in the domain of ARM refers to data stream and control of sensors [3]. The relationship between data mining and other artificial intelligence involving machines and network sensors is extendable, along with increasing opportunities [23]. Data representation and coverage may be insufficient when the mined knowledge is limited. Hence, the discovered knowledge typically encloses a set of rules. One feature of the discovered knowledge is that it should retain its validity when the database experiences nil change. However, the question that arises is: 'How does one extend the validity of knowledge with less processing time?' Precisely, knowledge represents the present state of the processed data with limited change in the data, inclusive of added, updated, or deleted transactions. The two commonly used approaches to maintain the legitimacy of knowledge are:

1. Rediscovery of knowledge from scratch upon changed state of the database.
2. Consideration of the impact of change on the database in terms of knowledge.

Upon database manipulation (adding, deleting, or updating transactions), the changes should be reflected in the elicited knowledge. While Apriori is the most common approach for ARM, other prominent algorithms do exist, such as Eclat [24], DHP [7], AprioriTID [25], McEclat [26], and MsApriori [5], which enhance collection of rare frequent items. Generally, the ARM process line mainly refers to the following two processes [13]:

1. The discovered large itemsets, also known as frequent itemsets, have more occurrences in the database than a pre-defined confidence threshold.
2. The frequent itemset generates strong association rules. A rule is considered strong and serves as output list if it satisfies the pre-defined confidence threshold.

Despite the existence of numerous techniques, ARM is the most widely used method for data mining. The analysis of market basket appears to be the most popular case study for association rules. The items in a market database are saved in a single record (transaction), along with customer ID, transaction time, and other imminent details. The discovering ARM includes rules, such as follows: let us assume a customer takes products x and y , but he is highly expected to purchase product w ; where x , y , and w are initially unknown. The customized promotional marketing program can be designed based on useful discovery rules. Apart from devising effective marketing plans, the data mining methods can be applied to both control and minimize consumption, such as energy and carbon dioxide (see [15,27]). The Apriori algorithm can effectively extract a frequent itemset from a massive dataset. Based on the discovered itemset on certain step (n), the algorithm generates an itemset called candidate itemset in the next step ($n + 1$) [2]. In each step, the algorithm checks the support counting for the candidate itemset using a vital scan database. The itemset generation process in a

level-by-level manner leads to level-wise problem. In fact, many algorithms have been introduced to address the multiple defects found in Apriori algorithm [10,24]. These algorithms are aimed at minimizing mining costs, apart from generating consistent and stable outcomes. In order to meet this goal, vast amount of data should be gathered and analyzed. One of the many mining issues refers to database size, as new transactions are dynamically updated and removed frequently. The already discovered rule sets need to be updated, or the rule set would not reflect the database. As such, maintaining ARM result incrementally has been examined in this study by placing focus on vertical mining. The proposed algorithm can overcome several existing drawbacks by reducing mining time and incorporating all manipulative operations. Minimizing database scan frequency is one of the proposed algorithm objectives. First, the algorithm builds a complete itemset, and later updates the set without rebuilding it to hinder level-wise issue. The paper is organized as follows. In the following section, the related terms are listed and defined. Section 3 presents the review of some past studies. In Section 4, the incremental Apriori (IApriori) is elaborated, and Section 5 describes the dataset. Section 6 presents the empirical outcomes and the comparison of the findings is depicted in Section 7.

2. Association Rule Terms

This section introduces and defines the important key terms used in ARM domain and the approach proposed in this study. The market basket was applied to clarify the terms and to illustrate the meaning. All the terms employed in this study are defined as follows:

Definition 1. The ‘set of items’ was given in the following:

$$I = \{i_1, i_2, i_3, \dots, i_m\}$$

where i refers to the attribute in the database. In market basket analysis, it denotes the product. It is a chance for the set of items to function as the range pool of attributes value in the database transaction. Based on the case study of ARM, the set of items refers to product found in the supermarket. Grounded on Definition 1, the term ‘transaction’ is defined as follows in the next definition:

Definition 2. The transaction, T : It is known as the main element of the database with a pair structure: $T = (tid, I')$, Where tid is the transaction identifier, while I' is a subset of I . The purchase of a set product by a customer is stored on a record or transaction market database. This operation is composed of two essential aspects; the ID for this record, and the items bought (I').

From Definition 2, the ‘transaction database’ is defined as follows:

Definition 3. Transaction database (DB) refers to a collection of transactions $\{T\}$ over I . DB is the source database that is scanned for the first time to generate new knowledge. From Definition 3, a special definition is derived for the modified database.

Definition 4. The modified transaction database ($db+$) is a collection of transactions $\{T\}$ over I , whereby $\{T\}$ is manipulated after mining DB. Simply put, the new transactions are transactions that have been either updated or deleted. Based on Definitions 1–4, the term ‘itemset’ is given in the following:

Definition 5. Itemset In the core point of ARM, the itemset is characterized as $x = \{i_1, i_2, i_3, \dots, i_m\}$, where x refers to a set of items that is supposed to befall frequently in a database as a pattern. The itemset has several features with ‘support’ being the most important one, defined as follows:

Definition 6 (itemset support). The itemset x support denotes the counting of T in D that contains the itemset x , such as:

$$\text{Support}(x, D) = |\text{tid} | (tid, i') D, X \subseteq I', I \subseteq I'|$$

Further clarification regarding the term ‘itemset support’ is related in the following example:

Example 1. If $x = i_1, i_2, i_3$ is an itemset in a D database that has 10 rows and x can be found in four transactions, then we have the following support of x :

$$\text{Support}(x) = 4/10 = 0.4$$

Based on Definitions 5 and 6, ‘support threshold’ is described as given in the following:

Definition 7. Support threshold (MinSupp) is the number or the ratio of itemset occurrence. If the itemset set has occurrence count exceeding α , it is considered as a frequent itemset. Nonetheless, the value of α is $0 \leq \alpha(x) \leq |D|$, with $|D|$ reflecting the database size. $\alpha(x)$ refers to the count of T that contains x itemset in the item set.

Definition 8. $X \Rightarrow Y$ represents the association rule form, where X, Y and $X \cap Y = \emptyset$. X is the body or the antecedent, whereas Y is the head or the consequence of the rule.

Definition 9. The confidence of an association rule $X \Rightarrow Y$ is the conditional probability of having Y embedded in a transaction, given that X is contained in that transaction.

$$\text{Confidence}(X \Rightarrow Y, D) = \text{Supprt}(X \cup Y, D) / \text{Support}(X, D)$$

The concept of association rule confidence is drawn in the following example.

Example 2. Let $D = T$ be a database of transactions. If a, b, c are itemsets, then the rule $a, b \cup c$ in D has the following confidence:

$$\text{Confidence}(a, b \Rightarrow c, D) = (\text{Supprt}(a, b, c, D)) / (\text{Support}(a, b, D))$$

The above definitions are linked with the ARM technique, and they are useful for most ARM algorithms. The following terms and definitions are applied to describe the proposed approach in this study.

Definition 10. Log file (Log F) states the changes that occur in a database. Generally, the database management system has a log file that states all actions that happen in the database. In the proposed method, three attributes are necessary to keep track of the database; transaction ID, action ID, and action date. Based on Definition 11, data mining is required to collect the target database.

Definition 11. The knowledge date (KD) stores the last mining date. For the next mining, KD is used to collect all manipulated transaction IDs, which were altered after the last mining. Based on the above two definitions, the following terms have been derived.

Definition 12. LTA is the list of transaction IDs that were added to the database (db+) after the KD date.

Definition 13. LTU is the list of transaction IDs that were modified in the DB after the KD date.

Definition 14. LTD is the list of transaction IDs that were deleted from the DB after the KD date.

Definition 15. The list of all item sets, L_n , in the proposed method, whereby the structure of the itemset has changed, and it is defined as a set: $L_n = \text{itemset}, \text{Support}, \text{tid}$ where itemset is a subset from I , support refers to support for the itemset, tid is the list of transaction IDs that contain the itemset, and n represents the length of the list.

Definition 16. Candidate set (C_n) refers to the intermediate list that combines an item in $L_{(n-1)}$.

Definition 17. Frequent item set (L_f) is a subset of (L_n), which contains all itemsets that satisfy the specific *MinSupp*. Section 3 presents the literature review, along with the criticism related to the existing approaches.

3. Literature Review

This section introduces the review of related research studies within two sub-sections. The first sub-section describes the Apriori algorithm, which is one of the leading approaches in the ARM domain. Next, the second sub-section depicts the incremental problem.

3.1. Apriori Algorithm

The Apriori algorithm is an icon algorithm in the ARM field [2,13]. Basically, this algorithm is employed to collect the frequent itemset based on pre-defined minimal support threshold. All parts (subsets) of the frequent itemset are frequent, whereby the itemset is considered frequent by the algorithm only when all subsets are frequent; otherwise the itemset is considered infrequent. The frequent pattern discovery process is described as an iterative process [28]. At the start of the iteration process, the algorithm counts the 1-itemsets. The algorithm seeks for the itemsets within the 1-itemsets to generate candidate 2-itemsets. The algorithm retrieves the database to count the support of 2-itemsets. Subsequently, the algorithm moves on to 3-itemsets, 4-itemsets, and K-itemsets. The algorithm derives a k-itemset candidate itemset from a k-1-itemset frequent itemset. Since the Apriori algorithm adheres to the downward closure lemma, all the candidate itemset subsets should be frequent. Generation of candidate sets continues until it is impossible to generate a new candidate itemset. Another essential algorithm for ARM is FP-growth [29]. The FP-growth determines both growth pattern and data structure based on FP-tree, which varies from the approach of Apriori that applies itemset data structure [29,30]. One of the many strategies on pattern discovery refers to the search for frequent patterns without specific support threshold, as employed in ARMGA [31] and G3PARM [32]. This strategy addresses one of the long-time standoff issues, which is support threshold parameter. However, this strategy is time consuming as it enters a specific threshold for every itemset to generate the rule, as well as to calculate the confidence for the rule and the support for the itemset. The next sub-section presents a review for incremental approaches.

3.2. Vertical Layout

The data layout in the database application is in horizontal format; this is because every record possesses numerous features that are not usually applied in a descriptive data mining process. Apriori TID [2] presented the vertical layout, where the data layout was changed from the horizontal to vertical layout, subsequently the vertical layout was applied to solved different issues in data mining field [24,26]. The layout employed intersection between the transaction ID lists to compute the support of a candidate's itemset. However, the algorithm proposed in the present research is an improvement because it reduces the number of times a database is scanned. In the proposed algorithm, a database is covered to only represent transactions in a vertical format in a TID list. Hence, mining efficiency is increased.

3.3. Incremental Problem

Data within a database are bound to change continuously. Simultaneously, the knowledge reflects the state of the database at the time of mining. Changes that take place in the data signify the losing potential for the knowledge to reflect the database. Therefore, updating discovered knowledge is an important task to ascertain that knowledge continues to reflect the state of the database. This strategy of updating the existing knowledge is called incremental learning within the search domain. Although a substantial number of methods have been proposed to overcome this incremental issue, they tend to display certain shortcomings, including time consuming, rebuilding knowledge, and selective

manipulation operations. As the databases are manipulated by time, the mining process becomes more complex than the mining of static dataset. Several problems may arise due to changes in the database, such as the generation of candidate itemset, and the scanning of the original database to validate the frequency of an itemset after database manipulation. The primary purpose of an incremental approach is to update the existing knowledge, so as to reflect the updated database state. This process should be carried out without the need to rebuild knowledge from scratch and without need to scan the entire database.

Fast Update Algorithm (FUP): Cheung, Han, Ng, and Wong (1996) initiated the first algorithm in the incremental ARM field area, known as fast update algorithm (FUP) [33]. This algorithm efficiently updates the existing association knowledge from the updated database. The *FUP* is similar to the Apriori algorithm, and it enhances the Apriori in generating a solution for issues related to incremental mining. Nevertheless, this *FUP* algorithm processes only recently added transactions on the manipulated database. Simply put, assume *DB* as the original database that requires data mining, while database with transactions added to it after mining is marked as *db*. Now, *DB+* refers to both *db* and *DB*. The *FUP* algorithm mines only *db*. Consider *x* is an itemset, so *x* has a chance to be frequent or perhaps not in one or both of the *DB* and *db* databases. In this case, as tabulated in Table 1, *x* has four possibilities.

Table 1. The possibilities of itemset over *DB* and *db*.

<i>DB</i> \ <i>db</i>	Frequent	Infrequent
Frequent	1—Frequent	2—Need to be check
Infrequent	3—Need to be check	4—Infrequent

As shown in Table 1, the itemset has four possibilities for an *x* itemset to be frequent. In the first case (1-Frequent), *x* is frequent in both *DB* and *db*, while in case 4 (4-Infrequent), *x* is infrequent in both databases; signifying nil issue for both cases. As for the other two cases, the *FUP* has to determine the support for the itemset over the entire database. The worst case is block 3, where the algorithm has to rescan the entire database to ascertain if the itemset is frequent or infrequent throughout the databases. Here, the *FUP* does not reduce the database scanning frequency. In the initial pass, the *FUP* scans the *db* to calculate the support count of each 1-itemset. After determining the support count of itemset in *DB*, the support of an itemset *x* is easily calculated if *x* is in block 2. If *x* appears in the third set, the whole *DB* must be scanned again to count the support of item *x*. The frequency of scanning cannot be reduced in *FUP* because all *DB*s demand multiple rescanning. Many studies have looked into the issue of discovery of ARM incrementally. One approach refers to *NFUP* [34] that serves to enhance the *FUP* algorithm. However, it targets only new transactions. The *IMSC* [35] is another approach that is meant to improve the *FUP* algorithm by addressing the constantly changing support threshold. The popular techniques in light of ARM algorithms, particularly Apriori-like algorithms, reflect downward closure in building an itemset. The *MAAP* [36], however, takes a different approach to build the frequent itemset. Where most ARM algorithms apply the bottom-up level, the *MAAP* starts from the top level, and based on definition, all subsets of the long frequent itemset become frequent as well. It is essential to develop an ARM algorithm that minimizes the number of database scans. This is a consequence of spending too much time while mining knowledge. The next section elaborates the proposed algorithm in detail, along with an example.

4. The Proposed Algorithm

The proposed algorithm, incremental Apriori (IApriori), is described in this section. The incremental learning approaches are explained in the previous sections. The defects of incremental algorithms were identified from the literature review. Apparently, two manipulation operations have been omitted during incremental learning, namely update and delete. These defects, thus, have

motivated this study to propose a method that embeds all manipulation operations when knowledge is learnt incrementally. A large size of itemsets are extracted from the transactions of the database by the Apriori algorithm, with each element in the itemsets having real support value. However, the expected size for itemset generation reflects an exponential growth to the size of the list of attributes in the database, thus it is impractical to count every subset found in the database transactions. The generation of candidate itemset is an iterative process with a combinatorial method that causes explosion on the itemset size (n). Reducing the frequency of database scan increases the algorithm performance, while increment in scanning frequency decreases the algorithm efficiency. One solution is to generate measured itemsets so that support can be counted earlier. Association rules are determined using three main steps in IApriori algorithm. First, the algorithm gathers all itemsets (L_n), where the length of each item is equal to one item and has support greater than zero. The next step is to combine the items in (L_n) to produce the all-possible frequent itemset (L_f) based on user requirements. Lastly, the association rule is generated only after creating the frequent itemset. The itemset structure created by IApriori is depicted in definition 15 ($L_n = \text{itemset}, \text{Support}, \text{tid}$). Each element in the L_n list has three attributes, namely itemset, support, and transaction ID list.

The main body of the IApriori algorithm is presented in Algorithm 1 pseudocode. The first line reads the last obtained result, while the second line categorizes the transaction IDs into three categories using the Proper Transaction List method. Note that these transaction IDs have been manipulated after the last mining. This particular method has many parameters: two and three as input and output parameters, respectively. As for the input parameters, the first is KD that refers to the last mining date, and the second is log file. This file contains every manipulation event over the transaction. Each record in the log file has three essential attributes: transaction ID, action ID, and time. A sample of the log file given in Table 2 shows that transaction ID refers to the manipulated transaction ID. The action ID is a code that reflects the manipulation operation. The output of the Proper Transaction List refers to three lists (LTA, LTU, and LTD) described in Definitions 13, 14, and 15, respectively. The process prior to mining new data should start after discarding the effects of deleted and updated transactions. The discard transaction isolates the effect of removed transactions, as presented in Algorithm 2. The preview instructions point out the specific target of the database for mining. The following describes the generation of the first itemset from new changes in the database, as illustrated in Algorithm 3. First, IApriori fetches the old discovered knowledge (if it exists) and places it in a list. The algorithm collects the target database transactions and formats them in the list. Next, two scenarios are bound to occur depending on the final outcome. In the first scenario, the presence of old result indicates that the algorithm has to update the value of the itemset attribute in the list. Otherwise, the entire itemset must be initialized and built from scratch. As for the second scenario, both the (generate candidates) and (get itemset) methods are applied, as given in Algorithms 4 and 5, respectively. Otherwise, the (update L_n itemset) is executed, as detailed in Algorithm 7. The algorithm should have a full itemset in this step. Based on the ARM concept, threshold is required. Hence, the next step is to apply *minsupp* in the get frequent itemset, as portrayed in Algorithm 8, to isolate all the itemsets that satisfy the *Minsupp* threshold. Finally, as a result of the previous step, the algorithm can generate the final output of the entire process, which is the association rule set.

Algorithm 1: IApriori Main Body.

```

 $L_n \leftarrow read(L_n)$ 
 $ProperTransactionList(LTA, LTU, LTD, LogF, KD)$ 
 $DiscardTransaction(LTU, LTD)$ 
 $L_1 \leftarrow GetL_1Items();$ 
If  $L_n = Null$ 
do
{
 $C_L \leftarrow GenerateCandidate(L_n);$ 
 $L_n \leftarrow GetItemset(C_L);$ 
}
while  $C_L \neq Empty$ 
else
 $updateL_nItemset(L_n, L_1);$ 
 $L_f \leftarrow GetFrequentset(Minsup, L_n);$ 
 $rule \leftarrow StrongRules(Minconf, L_f);$ 

```

Algorithm 2: IApriori- Discard Transaction Function.

```

ForEach ( $item \in L_n$ )
ForEach ( $TID \in (LTU, LTD)$ )
If ( $item.transaction.exist(TID)$ )
{
 $item.support - -;$ 
 $item.transaction.remove(TID);$ 
}

```

Algorithm 3: IApriori- $GetL_1ItemsFunction$.

```

 $Itemset \leftarrow Database.GetItem();$ 
ForEach ( $Item \in Itemset$ )
{
 $Item.Transaction \leftarrow Database.Read(tidcontainsItem);$ 
 $L_1.merge(item);$ 
 $Item.transaction.remove(tid);$ 
}

```

Algorithm 4: IApriori- Generate candidates function.

```

ForEach ( $Item \in L_n$ )
{
 $Candidateitem.itemset = L_n[Item.Location].itemset \cup L_n[Item.Location + 1].itemset$ 
 $Candidateitem.Transaction = L_n[Item.Location].TList \cup L_n[Item.Location + 1].TList$ 
 $Candidateitem.support = L_n[Item.Location].TList \cup L_n[Item.Location + 1].TList$ 
 $C_L.add(candidateitem);$ 
}

```

Algorithm 5: IApriori—get itemset function.

```

ForEach(Item in  $C_L$ )
  IF(Item.support > 0)
     $L_{n+1}.merge(item)$ ;
  
```

One fundamental method for the proposed algorithm is the Proper Transaction List, as displayed in Algorithm 6. This method gathers the transaction ID, which has been manipulated. The data for this method derive from the log file of the database management system (Log F) and the KD, which signify the latest date for updating the knowledge. This method generates three transaction ID lists for the transactions that have been manipulated after the KD date. The first list gathers the transactions IDs that have been added to the mined database (LTA), while the second list updates the transactions IDs (LTU), and the third list deletes transactions IDs (LTD). All log F records are collected using this method, and the transactions IDs are categorized on the list based on the manipulation operation type stored in the log file for each record.

Algorithm 6: IApriori—proper transaction list function.

```

OpenFile(LogF);
ForEach(row in LogF | row.date > KD)
  if(row.actionType = add)
    LTA.add(row.TID);
  Elseif(row.actionType = update)
    LTU.add(row.TID);
  Elseif(row.actionType = Del)
    LTD.add(row.TID);
  
```

Table 2. Set of rows from the database log file.

Date	Action ID	TID
02/01/2018 12:35	3	t_3
02/01/2018 12:40	2	t_4
29/12/2018 12:42	1	t_5
02/01/2018 12:52	1	t_6

Action ID: 1 Add, 2 Update, 3 Delete

Table 2 presents the sample from log file records. The latest changes on the database can be identified by using this log file. Upon detecting the changes made to the transactions, it facilitates mining to target the affected transactions. Typically, a log file is built in the management system to monitor changes that occur in the database. This database management system registers any operation that takes place in the database. Otherwise, a log file is created.

Algorithm 2 shows a vital feature of IApriori, which is the discard transaction method. The importance of this function stems from eliminating the manipulated transaction effect on the result obtained earlier. The effects of both deleted transactions (LTD) and updated transactions (LTU) are excluded from the result. The updated transactions are trained again with the added transaction to weigh in their new effect. The code for handling the excluded effect is given in Algorithm 2 pseudocode, whereby the method gathers the itemsets in L_n and determines if each item has one of the IDs of LTU and LTD in the itemset transactions list.

Algorithm 3 presents instructions of the $GetL_1$ item method. As the primary step in IApriori, the algorithm converts the present transaction to vertical layout view, thus changing the form of the items in the transaction to where the items occur in the database transactions. Clearly, the figure displays three attributes, namely the item element itself, the transaction IDs list, and the support of

the itemset. Algorithm 4 shows the generate candidates method in the proposed approach. The main variance between this method and the equivalent enhancement algorithm method lies in the pruning strategy. The IApriori offers the itemset some real support until the end of the generation process, due to two reasons: (1) the algorithm processes incremental learning so that the itemset can gain additional support in the next training and cumulate with the old one; and (2) the algorithm gives a chance for the users to filter the intermediate itemsets based on any threshold parameter without rebuilding knowledge from scratch. Despite its advantages, it is yet to be implemented as it has to filter the whole itemset based on specific attribute value to check if the attribute is linked with other attributes, regardless of the actual support value. Another vital aspect of the proposed approach is that the algorithm avoids infrequent issue (see Cases 2 and 3 in Table 1).

The IApriori considers all real itemsets with support value greater than zero. In Algorithm 5, this function isolates itemset that have never occurred in the database. Any itemset with a chance in the database is included in the next level of generating candidate itemsets.

The IApriori has its own characteristics, with one of them stated in Algorithm 7 pseudocode. It introduces the $updateL_N$ itemset method, which updates the whole intermediate itemset at once. This method is based on Algorithm 3, for the incremental training to gather the latest update from the database. After updating the first itemset list, this method performs intersection of the first itemset elements. For instance, x is an itemset that consists of three items $i_1i_2i_3$, and in order to collect both x support and tid, the algorithm intersects the tid for the items from $L_1 : i_1, i_2,$ and i_3 . From the L_1 itemset, the algorithm can maintain that L_n is collected in parallel.

Algorithm 7: IApriori—update L_n itemset function.

```
List < ID > IDNew
ForEach(m in  $C_L$ )
{
  IDNew =  $L_1.itemset[1].TID \cap \dots \cap L_1[m.itemset[n].TID$ ;
  m.TID.merge(IDNew);
  m.support=m.TID.count();
}
```

Algorithm 8 presents the GetFrequentItemset function, where many of the frequent itemsets are driven by this method based on the input threshold without the need to rebuild the itemset. Algorithm 9 shows the generation of strong rule method that is completely identical to the original method without making any change. Example 3 describes the proposed algorithm based on dual sections: the first phase demonstrates the creating and building of the intermediate itemset from the starting point, while the second phase displays the process of updating the incremental training.

Example 3. As presented in Table 3, a database contains four transactions. There is no obtained knowledge from this database. The demand has discovered the relationships between the items and found all the itemsets. The mining parameters; *MinSupp* and *Miscond*, are 50% and 60%, respectively. Table 4 shows the log file of the database and the action that occurred in the database transaction.

Algorithm 8: IApriori—get frequent itemset function.

```
ForEach(Item in  $L_n$ )
IF(Item.support  $\geq$  MinSupp)
   $L_f.add$ (Item);
```

Algorithm 9: IApriori- Get Strong Rules Function.

```

ForEach(Item in  $L_f$ ) {
  LHS=item.subList(0,item.lenght-1);
  RHS=item.subList(item.lenght-1,item.lenght);
  Rule = LHS  $\Rightarrow$  RHS;
  Rule.confidence = item.support / RHS.support;
  IF (Rule.confidence  $\geq$  MiConf)
  RuleList.add(Rule);
}

```

Table 3. A sample for database’s transaction.

TID	Items
t_1	ab
t_2	bc
t_3	ac
t_4	abc

Table 4. Log File 1.

TID	Action ID	Items
t_1	1	2017/06/18 12:30
t_2	1	2017/06/18 12:30
t_3	1	2017/07/09 12:40
t_4	1	2017/07/15 12:42

The main goal for the IApriori is to update the existing knowledge, thus the algorithm should possess the ability to read current knowledge. Since this is the very first time the algorithm is applied in a database, the algorithm built the knowledge from scratch. Typically, the absence of old result leads to the step where manipulated transactions are discarded. The following task prepares the transaction ID lists (LTA , LTU , and LTD), where LTA is $\{t_1, t_2, t_3, t_4\}$, LTU , and LTD are empty. However, under any case, the next step is to generate the first itemset by applying the $GETL_1$ items method. Table 5 shows the outputs of this step.

Table 5. Getting L1 by IApriori.

List Index	Itemset	Support	< tid >
1	{a}	3	$\{t_1, t_3, t_4\}$
2	{b}	3	$\{t_1, t_2, t_4\}$
3	{c}	3	$\{t_2, t_3, t_4\}$

After the first intermediate itemset has been built, the complete intermediate itemset is generated. Generating the whole intermediate itemset in this case refers to creating due to empty obtained knowledge. In the next level, the itemset has a size 2. The C_2 is $\{ab, ac, bc\}$. The support of C_2 is calculated by intersecting the first itemset, as given in the following:

$$ab.TID = a.TID \cup b.TID = \{t_1, t_3, t_4\} \cup \{t_1, t_2, t_4\}, \text{ then support of } \{ab\} \text{ is } 2.$$

$$ac.TID = a.TID \cup c.TID = \{t_1, t_3, t_4\} \cup \{t_2, t_3, t_4\}, \text{ then support of } \{ac\} \text{ is } 2.$$

$$bc.TID = b.TID \cup c.TID = \{t_1, t_2, t_4\} \cup \{t_2, t_3, t_4\}, \text{ then support of } \{bc\} \text{ is } 2.$$

Table 6 shows L_2 , along with support and TID. After completing the generation of L_2 , the IApriori moves on to create L_3 , whereby the creation and collection of the actual support are made by crossing the TID list. It is important to highlight that L_3 is limited to $\{abc\}$.

Table 6. Generate second itemset (L_2).

List Index	Itemset	Support	< tid >
1	{ab}	2	{t ₁ , t ₄ }
2	{ac}	2	{t ₃ , t ₄ }
3	{bc}	2	{t ₂ , t ₄ }

Table 7 shows the final and complete L_n itemset. At this end, the end-user can apply multiple parameters to arrive at vast knowledge, wherein this deriving process requires minimal time. For instance, based on the threshold *Minsupp* 50%, the frequent itemset is driven and is presented in Table 8.

Table 7. Complete itemset (intermediate itemset) (L_n).

List Index	Itemset	Support	< tid >
1	{a}	3	{t ₁ , t ₃ , t ₄ }
2	{b}	3	{t ₁ , t ₂ , t ₄ }
3	{c}	3	{t ₂ , t ₃ , t ₄ }
4	{ab}	2	{t ₁ , t ₄ }
5	{ac}	2	{t ₃ , t ₄ }
6	{bc}	2	{t ₂ , t ₄ }
7	{abc}	1	{t ₄ }

Table 8. Frequent itemset.

List Index	Itemset	Support	< tid >
1	{a}	3	{t ₁ , t ₃ , t ₄ }
2	{b}	3	{t ₁ , t ₂ , t ₄ }
3	{c}	3	{t ₂ , t ₃ , t ₄ }
4	{ab}	2	{t ₁ , t ₄ }
5	{ac}	2	{t ₃ , t ₄ }
6	{bc}	2	{t ₂ , t ₄ }

The association rules based on the frequent itemset presented in Table 8 are driven and listed in Table 9.

Table 9. Generated rule set.

List Index	Rule	Confidence
1	$a \Rightarrow b$	66.6%
2	$a \Rightarrow c$	66.6%
3	$b \Rightarrow a$	66.6%
4	$b \Rightarrow c$	66.6%
5	$c \Rightarrow a$	66.6%
6	$c \Rightarrow b$	66.6%

Finally, the mining process is completed for this example; the knowledge is built for the first time. In order to fully demonstrate the IApriori approach, the next example describes the incremental process. The subsequent example uses the latest example. Consider that the database in Table 10 is complete for the database presented in Table 3. With *MinSupp* parameter being 50%, the last result exists in the complete itemset list (L_n) (see Table 7). The tracked changes of the database are displayed in Table 11.

Table 10. An example of a transaction database.

TID	Items
t_4	ac
t_5	ab
t_6	abc

Table 11. Log File 1.

TID	Action ID	Date
t_3	3	2016/05/19 12:35
t_4	2	2016/05/19 12:40
t_5	1	2016/05/19 12:42
t_6	1	2016/05/20 12:42

Table 7 tabulates the final outcomes, whereby the algorithm reads the previous result. The next step is to collect $LTA, LTU, and LTD$; LTA is $\{t_5, t_6\}$, LTU is t_4 , and LTD is t_3 . The ensuing step discards the effects of LTD and LTU from the latest results. Table 12 presents the complete itemset after removing the deleted and updated transaction effects.

Table 12. Complete itemset L_n after discount support and remove TID in (LTA and LTU).

List Index	Itemset	Support	$\langle tid \rangle$
1	a	1	$\{t_1\}$
2	b	2	$\{t_1, t_2\}$
3	c	1	$\{t_2\}$
4	ab	1	$\{t_1\}$
5	ac	0	$\{\}$
6	bc	1	$\{t_2\}$
7	abc	0	$\{\}$

The IApriori targets both LTA and LTU to update the first itemset from the target transactions of the database, so as to generate an updated full itemset. $LTA \cap LTU = \{4, 5, 6\}$

The updated first itemset is presented in Table 13. Next, the IApriori updates the complete itemset. Table 14 tabulates the final fully updated itemset list.

Table 13. 1-itemsets (L_1).

List Index	Itemset	Support	$\langle tid \rangle$
1	a	3	$\{t_4, t_5, t_6\}$
2	b	2	$\{t_5, t_6\}$
3	c	2	$\{t_4, t_6\}$

Table 14. Final complete itemset L_n .

List Index	Itemset	Support	$\langle tid \rangle$
1	$\{a\}$	4	$\{t_1, t_4, t_5, t_6\}$
2	$\{b\}$	4	$\{t_1, t_2, t_5, t_6\}$
3	$\{c\}$	3	$\{t_2, t_4, t_6\}$
4	$\{ab\}$	3	$\{t_1, t_5, t_6\}$
5	$\{ac\}$	2	$\{t_4, t_6\}$
6	$\{bc\}$	2	$\{t_2, t_6\}$
7	$\{abc\}$	1	$\{t_6\}$

The previous step may derive numerous knowledge sets based on varying thresholds. The requirement in the example is $Minsupp = 50\%$ (three transactions). Based on that threshold value, the frequent itemset is listed in Table 15, while the final generated association rule for incremental example is presented in Table 16. The next section briefly describes the dataset.

Table 15. Frequent itemset L_f .

List Index	Itemset	Support	< tid >
1	{a}	4	{t ₁ , t ₄ , t ₅ , t ₆ }
2	{b}	4	{t ₁ , t ₂ , t ₅ , t ₆ }
3	{c}	3	{t ₂ , t ₄ , t ₆ }
4	{ab}	3	{t ₁ , t ₅ , t ₆ }

Table 16. Generated rule set.

List Index	Rule	Confidence
1	$a \Rightarrow b$	75%
2	$b \Rightarrow a$	75%

5. Dataset

The set of benchmark datasets were mined in numerous experiments to assess the proposed algorithm. The first was the Apriori dataset, which was evaluated using the proposed algorithm, wherein the ability to derive the same output in a specific threshold by the original Apriori algorithm was tested. It contained 75K records for supermarket database transactions with the dataset having 49 items. The dataset served as the benchmark dataset as it was produced by the Integrated and Project Management [37]. The second dataset refers to chess, whereby the dataset was obtained from the sports discipline with 3196 games and 36 attributes. Since the values of the attributes were categorized and the records had class attributes, they were suitable for classification mining. These were retrieved from fimi [38]. The third dataset refers to mushrooms, which contained imageries of hypothetical samples that matched 23 mushroom species designed specifically from mushroom hunters. Each item was labelled as edible, toxic, unknown, and unusable. The latter classification was combined with that of toxic. The dataset had 23 attributes and 8124 cases [38]. The dataset is available online at [38]. The fourth dataset refers to T10I4D100K dataset, which can be retrieved at [38]. The dataset is a market basket that contains 100K transactions with 1000 items. The fifth dataset is composed of breast cancer dataset, which was generated and published by the University Medical Center, Institute of Oncology. The breast cancer dataset contained 286 instances of medical cases. The dataset schema had nine attributes, which were split into linear and nominal attributes. The nominal attribute was categorized using the discretization method. The dataset is available online at [39]. Finally, the last dataset refers to the acute inflammations dataset, which resulted from the collaboration between the Laboratory of Intelligent Systems on Polish Academy of Sciences and medical expertise. The purpose of this dataset is to test the expert systems in performing a presumptive medical diagnosis for two diseases of the urinary system. This dataset had eight attributes, and 120 instances with each instance reflecting a patient’s case. The dataset is available at [39].

6. Results and Discussion

This section presents the empirical outputs derived from IApriori algorithm, along with comparison of the results with those obtained from baseline algorithm and AprioriTID. The experiments were performed to test and compare the scalability of the IApriori algorithms. The experimental work evaluated the changes noted in the threshold for the chess dataset. Table 17 and Figure 1 present the retrieved results. The knowledge from the output appeared similar for both algorithms. The IApriori displayed exceptional achievements in light of execution time, whereby

IApriori algorithm had successfully decreased the processing time between 25% and 50%. Meanwhile, the Apriori algorithm took between two and four times longer processing time to perform the same tasks. The results clearly exhibit the potential of IApriori algorithm in terms of speed, when compared to other algorithms. The time values portrayed in the table are in milliseconds.

Table 17. The result of experiments over the chess dataset (executing time).

Support	Apriori	AprioriTID	IApriori	Improvement (%)
0	699	702	705	0.64
1	675	502	170	246.18
2	635	328	49	882.65
3	575	389	49	883.67
4	585	309	66	577.27
5	575	339	58	687.93
6	525	419	75	529.33
7	485	389	80	446.25
8	475	339	59	589.83
9	465	319	90	335.56
10	462	309	100	285.50

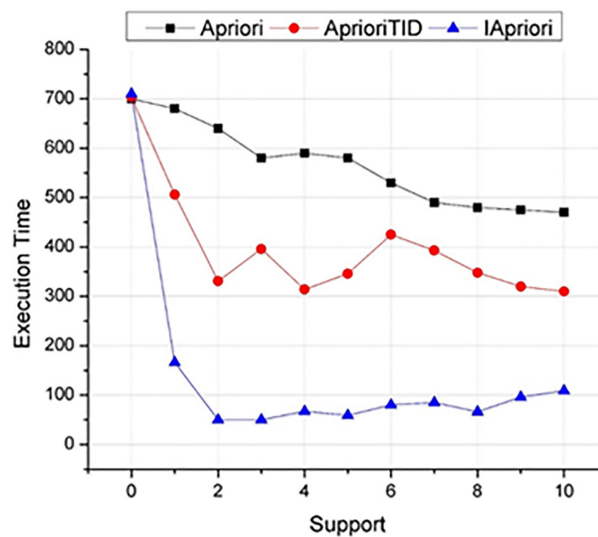


Figure 1. Experimental results for chess dataset (execution time).

The next experiment involved the mushroom dataset with under different values of support threshold, whereby outcomes retrieved from the two algorithms were compared. The IApriori algorithm displayed improvement for execution time from 15% to 80%. Table 18 and Figure 2, which present the empirical findings for the mushroom dataset, show that the improvement rates were rather high.

Table 18. The result of experiments over mushroom dataset (executing time).

Support	Apriori	AprioriTID	IApriori	Improvement (%)
0	632	801	518	38.32
1	709	526	70	782.14
2	655	617	58	996.55
3	488	591	57	846.49
4	578	466	59	784.75
5	539	488	61	741.80
6	426	434	69	523.19
7	253	362	72	327.08
8	306	399	65	442.31
9	415	395	70	478.57
10	351	264	69	345.65

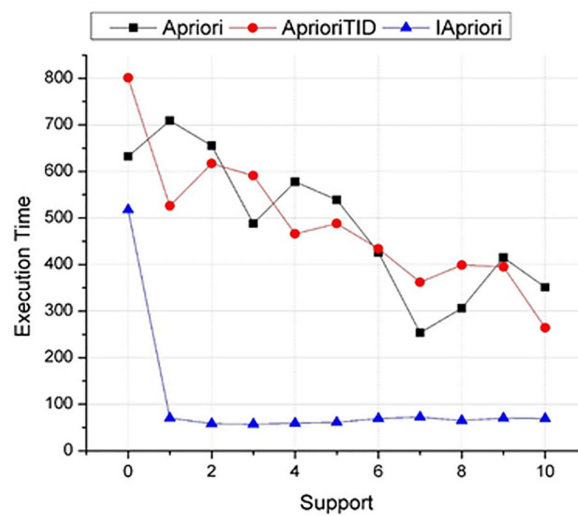


Figure 2. Experimental results for mushroom dataset (execution time).

The T10|4D100K dataset was used to test the IApriori algorithm, whereby the results were compared with those obtained from AprioriTID and Apriori. Figure 3 and Table 19 tabulate the experimental results involving T10|4D100K dataset. The results of both IApriori and AprioriTID algorithms appeared logical and reasonable, while the outcomes retrieved from Apriori took longer time. It is emphasized here that the experimental setting was retained similar for all algorithms.

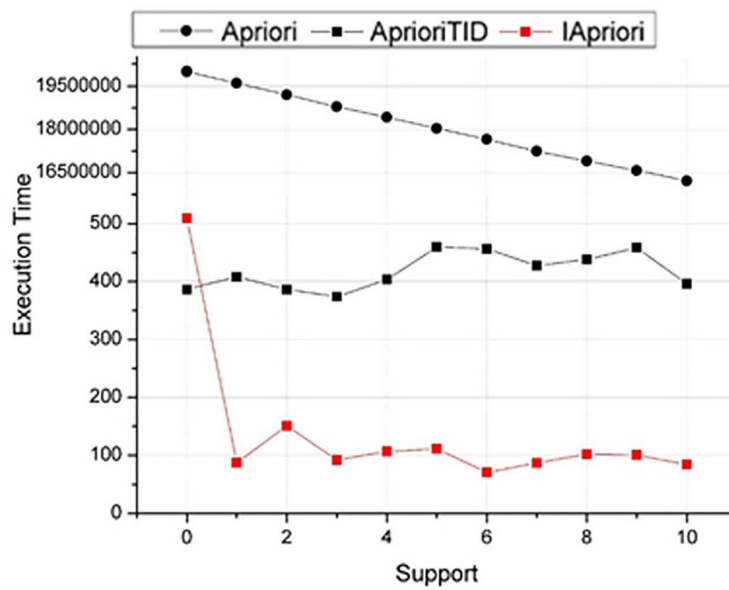


Figure 3. Experimental results for T10|4D100K dataset (execution time).

The following table presents the numerical outputs from experiments using the T10|4D100K dataset.

Table 19. The result of experiments over T10|4D100K Dataset (Executing Time).

Support	Apriori	AprioriTID	IApriori	Improvement (%)
0	20,000,000	386	509	1,964,574
1	19,604,746	408	87	11,267,230
2	19,205,401	386	151	6,359,432
3	18,785,613	374	92	10,209,676
4	18,423,390	404	107	8,609,150
5	18,031,557	460	111	8,122,430
6	17,654,125	456	70	12,610,315
7	17,240,452	427	87	9,908,451
8	16,901,257	438	102	8,285,045
9	16,571,963	458	100	8,286,111
10	16,212,692	396	84	9,650,548

Figure 4 clearly presents the recorded findings. Figure 4 portrays the comparison of results obtained from IApriori algorithm with those of other tested algorithms on the breast cancer dataset. Apparently, IApriori achieved the best execution time. For the first experiment (when threshold was zero), the IApriori demanded additional time, but was still less when compared to the other Apriori algorithms. The following experiments required no time less than 100 ms. For instance, when the support threshold was 4, IApriori recorded 95 ms, whereas 350 and 550 ms for AprioriTID and Apriori, respectively. This achievement reflects outstanding improvement up to 300%.

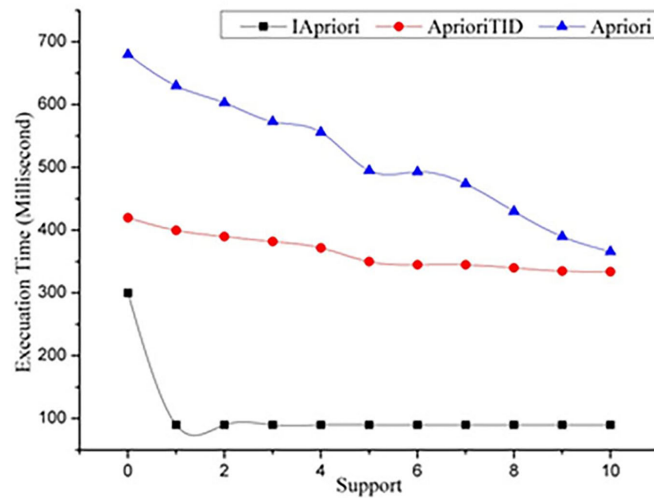


Figure 4. Experimental results for breast cancer dataset (execution time).

Figure 5 shows the experimental results obtained from the algorithms for Acute Inflammations dataset. Again, the IApriori displayed excellent execution time. The time required to mine the dataset by IApriori (support threshold = two) was 98 ms, while 395 and 605 ms for AprioriTID and Apriori, respectively.

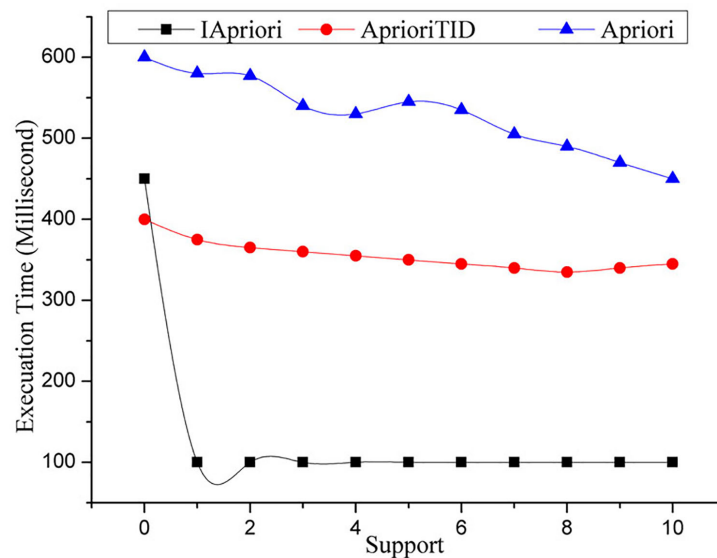


Figure 5. Experimental results for Acute Inflammations dataset (execution time).

The next experimental work captures the ability of the IApriori Algorithm to process different dataset sizes, as well as the effect of dataset size on execution time. Additionally, the IApriori incorporated the changing threshold. Figure 6 presents the outcomes of mining experiments with a set of database sizes (1K, 5K, 20K, and 75K) (Management, 2009). The execution time was about the average time for 10 experiments for each threshold value, whereby the support threshold changed from 0% to 10%. It appeared that addition time was required for the first experiment only, which was 710 ms, for the dataset with 75K transactions.

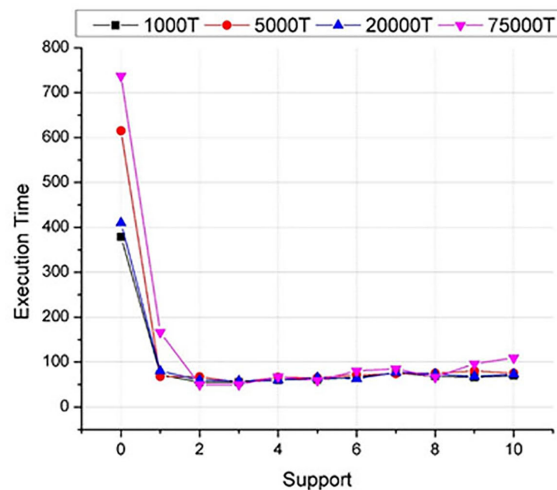


Figure 6. Execution time (ms) under different thresholds for the same transaction size (Apriori dataset).

The above figures portray the outstanding ability displayed by IApriori in substantially minimizing the execution time. This achievement was tested on various types of datasets across multidisciplinary domains, along with the standard case study for ARM, namely basket market analyses, medical, and game datasets mined by IApriori. The IApriori performed rather exceptionally well for different types and sizes of datasets, as well as for different values of support threshold parameter. Overall, IApriori has achieved remarkable outcomes.

6.1. Comparison

This section discusses the theoretical comparison between IApriori and other algorithms. This comparison is divided into two parts: IApriori is compared with the baseline algorithm in the first part, while on the incremental domain in the second part.

6.2. Apriori

The IApriori algorithm seemingly managed to enhance the Apriori algorithm, which is attributed to their shared similarities. The variances are as listed in the following:

- IApriori algorithm retrieved the data from the database once; it collected the first itemset. Subsequently, the algorithm collected the support count for the candidate itemset by intersecting the first itemset. The original algorithm had to check the database for every single itemset.
- The Apriori algorithm applied the pruning strategy to exclude itemsets that did not satisfy the *Minsupp* threshold. In the proposed method, the algorithm retained all itemsets.
- The Apriori algorithm built the whole knowledge-based over the specific threshold, and the output knowledge appeared to be invalid for other thresholds. Upon a change in the threshold, the mining had to restart from the beginning point of building valid knowledge. On the contrary, many instances can be arrived at using the proposed algorithm in the case of changing threshold.

6.3. Incremental Algorithms

Many algorithms have been proposed in the field of incremental learning for ARM. They make sense in the process of solving standoff problems. The method proposed in this study presents an improved set of features. However, several variances were noted, as listed below:

- IApriori weighed in all manipulated data found in the database, including added, updated, and deleted data. On the contrary, the remaining algorithms that dealt with the incremental issue considered only the add operation, while neglecting both update and delete operations.
- As for the other algorithms, there were cases in which the algorithm had to rebuild the itemset from scratch. This, nonetheless, was not required for the proposed algorithm.
- IApriori offered credible support and confidence for itemset and rule set, respectively. IApriori updated the support value, thus reflecting the itemset to the database transactions. Additionally, IApriori filtered the itemset within the changeable threshold setting.

The next table (Table 20) summarizes the features of the reviewed incremental ARM Algorithms.

Table 20. Comparison between the proposed method and incremental algorithms.

Algorithm	Add	Update	Delete	Rescan Solved	Support Accuracy	Threshold Changes
FUP [33]	✓	X	X	X	X	X
NFUP [34]	✓	X	X	X	X	X
IMSC [35]	✓	X	X	X	X	✓
MAAP [36]	✓	X	X	X	X	X
IApriori	✓	✓	✓	✓	✓	✓

7. Conclusions

This study has addressed the ARM incremental problem by enhancing the popular Apriori algorithm [2,13]. The ARM algorithms extract knowledge from the database, where the extracted knowledge reflects the database when no change is introduced. Indeed, the database status in real-time applications is changeable, and the database can be manipulated by numerous database operations (add, update, and delete). Many algorithms have been proposed to overcome this issue in the ARM incremental field. Nevertheless, several weaknesses and defects exist and remain, such as the following: the present approaches target, in the mining, the newly extended database only for those transactions that were added after the last mining, and they suppose, with the passage of time, that the state of the old database is constant and that no change has been made; neither deleted nor updated transactions. Based on Table 1, many cases demand rescanning all databases, particularly when there is a frequent itemset in the new database, but infrequent in the old database. In this case, scanning the database demands checking if the new itemsets are frequent over the entire database. Moreover, thresholds are one of the most popular problems in most learning techniques. In ARM, knowledge discovery relies on specific thresholds; so any changes in threshold indicate that the discovered knowledge does not reflect the database anymore. The proposed algorithm allows change in the thresholds and retrieves new knowledge based on the new thresholds without rescanning the database.

The proposed algorithm perfectly managed to overcome the previous outstanding issues, apart from providing effective and viable solutions for a range of domains. The IApriori algorithm was tested in many experiments by placing focus on two specific features: execution time, and the similarity of extracted knowledge between the proposed algorithm and the others, as results showed the improvement made by the IApriori on Tables 17–19 and Figures 1–5. This was performed to ascertain the accuracy attained in extracting knowledge.

Author Contributions: Methodology, software, and original draft preparation, I.A. Validation and supervision N.A.G. Editing C.M. Editing and review J.M. Review and testing N.S.S.

Funding: This research was funded by University Malaya through a postgraduate research grant (PPP) grant number PG106-2015B.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hashem, T.; Ahmed, C.F.; Samiullah, M.; Akther, S.; Jeong, B.S.; Jeon, S. An efficient approach for mining cross-level closed itemsets and minimal association rules using closed itemset lattices. *Expert Syst. Appl.* **2014**, *41*, 2914–2938. [[CrossRef](#)]
2. Agrawal, R.; Imieliński, T.; Swami, A. Mining association rules between sets of items in large databases. *Acm sigmod record. ACM* **1993**, *22*, 207–216.
3. Chen, Q.; Fan, Z.; Kaleshi, D.; Armour, S. Rule induction-based knowledge discovery for energy efficiency. *IEEE Access* **2015**, *3*, 1423–1436. [[CrossRef](#)]
4. Djenouri, Y.; Djenouri, D.; Belhadi, A.; Fournier-Viger, P.; Lin, J.C.W. A new framework for metaheuristic-based frequent itemset mining. *Appl. Intell.* **2018**, *48*, 4775–4791. [[CrossRef](#)]
5. Lee, Y.C.; Hong, T.P.; Lin, W.Y. Mining association rules with multiple minimum supports using maximum constraints. *Int. J. Approx. Reason.* **2005**, *40*, 44–54. [[CrossRef](#)]
6. Nguyen, D.; Vo, B.; Le, B. CCAR: An efficient method for mining class association rules with itemset constraints. *Eng. Appl. Artif. Intell.* **2015**, *37*, 115–124. [[CrossRef](#)]
7. Park, J.S.; Yu, P.S.; Chen, M.S. *Mining Association Rules With Adjustable Accuracy*; IBM Thomas J. Watson Research Division: Armonk, NY, USA, 1997.
8. Usman, M.; Usman, M. Multi-Level Mining and Visualization of Informative Association Rules. *J. Inf. Sci. Eng.* **2016**, *32*, 1061–1078.
9. Nguyen, L.T.; Nguyen, N.T.; Vo, B.; Nguyen, H.S. Efficient method for updating class association rules in dynamic datasets with record deletion. *Appl. Intell.* **2018**, *48*, 1491–1505. [[CrossRef](#)]
10. Li, W.; Han, J.; Pei, J. CMAR: Accurate and efficient classification based on multiple class-association rules. In Proceedings of the 2001 IEEE International Conference on Data Mining, Washington, DC, USA, 29 November–2 December 2001; pp. 369–376.
11. Lin, C.W.; Hong, T.P.; Lan, G.C.; Wong, J.W.; Lin, W.Y. Efficient updating of discovered high-utility itemsets for transaction deletion in dynamic databases. *Adv. Eng. Inform.* **2015**, *29*, 16–27. [[CrossRef](#)]
12. Soysal, Ö.M. Association rule mining with mostly associated sequential patterns. *Expert Syst. Appl.* **2015**, *42*, 2582–2592. [[CrossRef](#)]
13. Agrawal, R.; Srikant, R. Fast algorithms for mining association rules. In Proceedings of the 20th International Conference on Very Large Data Bases, VLDB, San Francisco, CA, USA, 12–15 September 1994; Volume 1215, pp. 487–499.
14. Ogbah, H.; Alashqur, A.; Qattous, H. Predicting Heart Disease by Means of Associative Classification. *Int. J. Comput. Sci. Netw. Secur. (IJCSNS)* **2016**, *16*, 24.
15. De Oliveira, E.F.; de Lima Tostes, M.E.; de Freitas, C.A.O.; Leite, J.C. Voltage thd analysis using knowledge discovery in databases with a decision tree classifier. *IEEE Access* **2017**, *6*, 1177–1188. [[CrossRef](#)]
16. Yassine, A.; Singh, S.; Alamri, A. Mining human activity patterns from smart home big data for health care applications. *IEEE Access* **2017**, *5*, 13131–13141. [[CrossRef](#)]
17. Lee, S.; Ryu, K.; Shin, M.; Cho, G.S. Function and service pattern analysis for facilitating the reconfiguration of collaboration systems. *Comput. Ind. Eng.* **2012**, *62*, 794–800. [[CrossRef](#)]
18. Bose, I.; Mahapatra, R.K. Business data mining—A machine learning perspective. *Inf. Manag.* **2001**, *39*, 211–225. [[CrossRef](#)]
19. Chen, F.; Wang, Y.; Li, M.; Wu, H.; Tian, J. Principal association mining: an efficient classification approach. *Knowl.-Based Syst.* **2014**, *67*, 16–25. [[CrossRef](#)]
20. Kumara, B.T.; Paik, I.; Siriweera, T.; Koswatte, K.R. Cluster-based web service recommendation. In Proceedings of the 2016 IEEE International Conference on Services Computing (SCC), San Francisco, CA, USA, 27 June–2 July 2016; pp. 348–355.
21. Rashid, M.M.; Gondal, I.; Kamruzzaman, J. Dependable large scale behavioral patterns mining from sensor data using Hadoop platform. *Inf. Sci.* **2017**, *379*, 128–145. [[CrossRef](#)]
22. Sheu, J.J.; Chen, Y.K.; Chu, K.T.; Tang, J.H.; Yang, W.P. An intelligent three-phase spam filtering method based on decision tree data mining. *Secur. Commun. Netw.* **2016**, *9*, 4013–4026. [[CrossRef](#)]
23. Gandhi, N.; Armstrong, L.J. A review of the application of data mining techniques for decision making in agriculture. In Proceedings of the 2016 2nd International Conference on Contemporary Computing and Informatics (IC3I), Noida, India, 14–17 December 2016; pp. 1–6.

24. Zaki, M.J.; Parthasarathy, S.; Ogihara, M.; Li, W. Parallel algorithms for discovery of association rules. *Data Min. Knowl. Discov.* **1997**, *1*, 343–373. [[CrossRef](#)]
25. Li, Z.C.; He, P.L.; Lei, M. A high efficient AprioriTid algorithm for mining association rule. In Proceedings of the 2005 International Conference on Machine Learning and Cybernetics, Guangzhou, China, 18–21 August 2005; Volume 3, pp. 1812–1815.
26. Schlegel, B.; Karnagel, T.; Kiefer, T.; Lehner, W. Scalable frequent itemset mining on many-core processors. In Proceedings of the Ninth International Workshop on Data Management on New Hardware, New York, NY, USA, 24 June 2013; p. 3.
27. Ge, Z.; Song, Z.; Ding, S.X.; Huang, B. Data mining and analytics in the process industry: The role of machine learning. *IEEE Access* **2017**, *5*, 20590–20616. [[CrossRef](#)]
28. Wu, X.; Fan, W.; Peng, J.; Zhang, K.; Yu, Y. Iterative sampling based frequent itemset mining for big data. *Int. J. Mach. Learn. Cybern.* **2015**, *6*, 875–882. [[CrossRef](#)]
29. Han, J.; Pei, J.; Yin, Y. Mining frequent patterns without candidate generation. In Proceedings of the 2000 ACM SIGMOD international conference on Management of Data, New York, NY, USA, 15–18 May 2000; Volume 29, pp. 1–12.
30. Ramya, V.; Ramakrishnan, M. FP-growth algorithm based incremental association rule mining algorithm for big data. *Int. J. Adv. Res. Comput. Sci.* **2018**, *9*, 886. [[CrossRef](#)]
31. Yan, X.; Zhang, C.; Zhang, S. Genetic algorithm-based strategy for identifying association rules without specifying actual minimum support. *Expert Syst. Appl.* **2009**, *36*, 3066–3076. [[CrossRef](#)]
32. Luna, J.M.; Romero, J.R.; Romero, C.; Ventura, S. Reducing gaps in quantitative association rules: A genetic programming free-parameter algorithm. *Integr. Comput.-Aided Eng.* **2014**, *21*, 321–337. [[CrossRef](#)]
33. Cheung, D.W.; Han, J.; Ng, V.T.; Wong, C. Maintenance of discovered association rules in large databases: An incremental updating technique. In Proceedings of the Twelfth International Conference on Data Engineering, New Orleans, LA, USA, 26 February–1 March 1996; pp. 106–114.
34. Chang, C.C.; Li, Y.C.; Lee, J.S. An efficient algorithm for incremental mining of association rules. In Proceedings of the 15th International Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications (RIDE-SDMA'05), Tokyo, Japan, 3–4 April 2005; pp. 3–10.
35. Bachtobji, M.A.; Gouider, M.S. Incremental maintenance of association rules under support threshold change. In Proceedings of the IADIS International Conference on Applied Computing, San Sebastian, Spain, 25–28 February 2006; IADIS Press: San Sebastian, Spain, 2006.
36. Zhou, Z.; Ezeife, C. A low-scan incremental association rule maintenance method based on the apriori property. In Proceedings of the Conference of the Canadian Society for Computational Studies of Intelligence, Ottawa, ON, Canada, 7–9 June 2001; Springer: Heidelberg, Germany, 2001; pp. 26–35.
37. Integrated & Project Management. Available online: <https://wiki.csc.calpoly.edu/datasets/wiki/apriori> (accessed on 15 October 2019).
38. Frequent Itemset Mining Dataset Repository. Available online: <http://fimi.ua.ac.be/data/> (accessed on 15 October 2019).
39. UC Irvine Machine Learning Repository. Available online: <https://archive.ics.uci.edu> (accessed on 15 October 2019).

