DRIMS: A Software Tool to Incrementally Maintain Previous Discovered Rules

Alain Pérez-Alonso[1](✉), Ignacio J. Blanco[2], Jose M. Serrano[3], and Luisa M. González-González[1]

[1] University "Marta Abreu" of Las Villas, 54830 Santa Clara, Villa Clara, Cuba
{apa,luisagon}@uclv.edu.cu
[2] University of Granada, 18071 Granada, Spain
iblanco@decsai.ugr.es
[3] University of Jaén, 23071 Jaén, Spain
jschica@ujaen.es

Abstract. A wide spectrum of methods for knowledge extraction have been proposed up to date. These expensive algorithms become inexact when new transactions are made into business data, an usual problem in real-world applications. The incremental maintenance methods arise to avoid reruns of those algorithms from scratch by reusing information that is systematically maintained. This paper introduces a software tool: Data Rules Incremental Maintenance System (DRIMS) which is a free tool written in Java for incrementally maintain three types of rules: association rules, approximate dependencies and fuzzy association rules. Several algorithms have been implemented in this tool for relational databases using their active resources. These algorithms are inspired in efficient computation of changes and do not include any mining technique. We operate on discovered rules in their final form and sustain measures of rules up-to-date, ready for real-time decision support. Algorithms are applied over a generic form of measures allowing the maintenance of a wide rules' metrics in an efficient way. DRIMS software tool do not discover new knowledge, it has been designed to efficiently maintain interesting information previously extracted.

Keywords: Association rules · Approximate dependencies · Incremental maintenance · Active databases

1 Introduction

Association Rules (ARs), Approximate Dependencies (ADs) and Fuzzy Association Rules (FARs) are ones of the best studied models for knowledge discovery in the data mining research field. They represent associations or dependencies among attributes' values in a data repository [1]. Many algorithms have been proposed to improve the mining process and create more efficient methods [4,22,35]. However, these proposed algorithms could become expensive when dealing with huge amounts of data, commonly stored in data warehouses or very large and big databases.

The knowledge discovered by these methods is specific for the current stage of the repository in which they were run. In real-world applications, a data repository is not static and records are commonly inserted, updated or deleted, following real activities in the universe of discourse. These continuous changes can render the measures of rules inexact and eventually invalid [16]. The incremental rule mining method arise to avoid re-run algorithms from scratch and re-scan the whole data. This is specifically useful when real-time data information is required. Example applications can be found in the field of data streams like web click stream data, sensor networks data, and network traffic data [19,20,32]. Emerging research in big data offers similar issues in association with velocity and volume [27,34]. At this time, many research efforts are being made to improve the performance [14,16,18].

In this work we describe a Data Rules Incremental Maintenance System (DRIMS) tool that formally implements incremental maintenance algorithms into relational databases using their active resources [24]. DRIMS is a free and open-source tool completely written in Java available from the GitHub platform [25]. There are two main applications of the tool: (1) the tool safety manages a repository of rules such as creating new rules, and (2) the tool can generate an SQL script to maintain the rules measures up-to-date in an efficient way. We also present the algorithms implemented in DRIMS, and the experimental results obtained from active relational database with real educational data and repository datasets. A common characteristic of algorithms implemented is the efficient maintenance of existing rules, keeping their measures just-in-time available for real-time decision support [29].

The remainder of this paper is organized as follows. Some related works are reviewed in Sect. 2. The algorithms implemented in DRIMS are described in Sect. 3. In Sect. 4 we briefly present the graphical user interface. Section 5 presents the experimental results of the proposed methods for the performance evaluation. Finally, Sect. 6 concludes this paper summarizing the results of our work.

2 Related Work

Association rules can formally be represented as an implication of itemsets (sets of items) in transactional databases [1]. Let $It = \{It_1, It_2, \ldots, It_m\}$ be a non-empty set of m distinct attributes. Let T be the transaction scheme that contains a set of items such that $It \subseteq T$. An AR is an implication of the form $X \Rightarrow Y$ where $X, Y \subset It$ such that $X \neq \emptyset$, $Y \neq \emptyset$ and $X \cap Y = \emptyset$. In this statement X and Y are called rule itemsets and they are the antecedent and consequent of the rule, respectively. The ADs and FARs can be represented through an AR perspective. We follow our research group's results in ADs [28] and FARs [9].

Numerous algorithms for mining ARs have been proposed at this time based on Apriori approach [1]. These Apriori-like algorithms generate the candidate itemsets level-by-level, which might cause multiple scans of the database and high computational costs. In order to avoid re-scanning the whole data and breaking Apriori bottlenecks, many algorithms have been proposed by using tree-structures [14,30]. The frequent-pattern tree (FP-tree) proposed by [12] is a milestone for the development of ARs based upon this method. The FP-tree is used to compress a database into a tree structure which stores only large items. After the FP-tree is constructed, a mining algorithm called FP-growth derives all large itemsets in a second step [12].

In real-world applications, data repository is not static. Generally, data will increase with time. Traditional batch mining algorithms solve this problem by re-scanning the whole data when new transactions are inserted, deleted or modified. This is clearly inefficient because all previous mined information is wasted. The incremental mining defines this issue as an update problem and reduces it to find the new set of large itemsets incrementally. Algorithm FUP (Fast UPdate) [8], is the first algorithm for incremental mining of ARs when new data transactions are added to a database.

Although the FUP approach improves a mining performance in dynamic environments, the original database is still required to be re-scanned. Extended tree structures are being designed for FP-tree to handle efficiently this problem [14,16,18]. These proposals improve the pioneer tree-structure in different ways but maintain the execution of FP-growth algorithm in a second step. Some related researches are still in progress.

Unlike incremental mining methods, DRIMS handles the update problem by maintaining the measures of previous discovered rules, just like we explain in detail for ARs and ADs [26]. That does not lead to maintain itemsets information, instead, existing rules measures are directly updated in an incremental way. After the rules discovering process, DRIMS keeps only the extracted rules. In Fig. 1 three scenarios are illustrated when a system decision-maker needs the

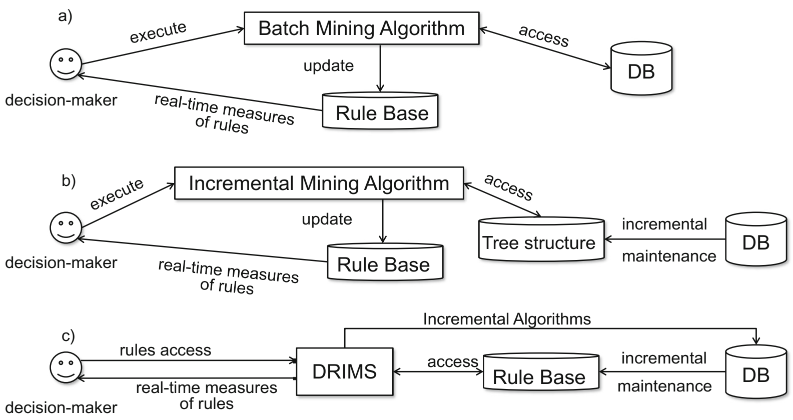


Fig. 1. (a) Real-time measures by batch mining method, (b) incremental mining method, and (c) DRIMS incremental maintenance proposal.

real-time measures of previously discovered rules. That includes the batch mining method, the incremental mining method, and DRIMS algorithms.

3 Data Rules Maintenance Proposals

ARs, ADs and FARs are different data relationships that share some similarities [21]. These data dependencies are referred to as Data Rules (DRs) in the remainder of this paper for a common reference. Many research activities propose measures of rules with different properties such as the certainty factor [3], and their number is overwhelming [10,15]. Existing measures for DRs are usually defined by counting a total number of records that satisfy some condition. These conditions are generally associated with the antecedent, consequent, rule examples, and counterexamples among others [10,15].

In DRIMS algorithms, each DR measure value is considered a set of k distinct data rule measure-parts $DRM = \{Mp_1, Mp_2, \ldots, Mp_k\}$ in which each item represents a different part of the measure formula. Measure-parts must be atomic, it means that they cannot be divided into smaller items and still bring the same measure value. For example, confidence can be split into two parts: count of antecedent and count of (antecedent $\cup$ consequent). On the other hand, the certainty factor needs three parts: count of antecedent, count of consequent and count of (antecedent $\cup$ consequent). In this way, it is possible to maintain efficiently several metrics at the same time because metrics shares some measure-parts. For example, following [15] it is viable with only five distinct measure-parts to maintain 20 interestingness measures simultaneously. The final data rule measure value is a formula over DRM parts.

3.1 Immediate Incremental Maintenance Algorithm

An immediate approach in DRIMS is oriented to update the rule base immediately after the event takes place, in an active fashion. The primitive event type, called a primitive structural event (PSE), is a single low-level event. A composite type is a combination of multiple primitive or composite structural events (CSE). This immediate approach verifies the specific rules that must be updated only with the changes made by a PSE. It means that only one record can be checked at a time. Incremental view maintenance algorithms offer multiple solutions. Specifically, a counting algorithm for view maintenance [11] provides an interesting perspective. The following Algorithm 1 presents the proposed immediate incremental maintenance where rule measures are updated for data operations.

The measure-parts Mp_k are constantly updated in the proposed algorithm. However, the incrementing of measure-parts are quite different depending on the DR type. For example, to obtain an AD measure each measure-part $Mp_k \in ADM$ is calculated by aggregating the *catt* attribute in AE_k [26]. For this immediate proposal, rule base refreshing is made without access to the base relation.

Algorithm 1. Immediate incremental maintenance for a DR

```
Input: A composite structural event $CSE$ that modifies the attributes related
       in list $L$, and measure-parts $DRM$ of $X \Rightarrow Y$ data rule.
Output: An updated measure-parts $DRM$.
Method:
1  foreach $PSE \in CSE$ do
2      if ($PSE = \Delta\ t^{-+}$) then                               /* update event */
3          if ($L \cap \{X \cup Y\} \neq \emptyset$) then
4              foreach $Mp_k \in DRM$ do
5                  if ($L \cap$ {involved attributes in $Mp_k$} $\neq \emptyset$) then
6                      update $Mp_k$, increment with $t_0^{-+}$;
7                      update $Mp_k$, decrement with $t_1^{-+}$;
8      else if ($PSE = \Delta\ t^{+}$) then                             /* insert event */
9          foreach $Mp_k \in DRM$ do
10             update $Mp_k$, increment with $t^{+}$;
11     else                                          /* delete event ($PSE$ = $\Delta t^{-}$) */
12         foreach $Mp_k \in DRM$ do
13             update $Mp_k$, decrement with $t^{-}$;
14 return $DRM$
```

3.2 Deferred Incremental Maintenance Method

A deferred approach efficiently maintains a fuzzy rule base up-to-date but not for each data operation. This method computes modified instances in a data transition and updates fuzzy rule base for these relevant instances. Principal differences of immediate and deferred maintenance approaches are illustrated in Fig. 1.

The deferred proposal is divided into two subproblems. The first subproblem consists of computing the relevant affected instances of a database transition. In this step, we build a relevant operations set at real-time, after each primitive structural event takes place. A different approach would be to scan the original operation set to reduce their number. The second one is related to incrementally update the rule base with those relevant instances.

Affected transition instances computation must consider the relationships among primitive events. These interactions are controlled by net effect policies [6,7,24]. For example, if a record is inserted and next deleted in the same database transition, then these events do not provoke any variation to the final database state and its measures of rules. In this approach, each database relation related with any rule has two auxiliary relations: insert relation (I) and delete relation (D). The insert and delete auxiliary relations register the insert, update, and delete events ($t^{+} \cup t_1^{-+}$ and $t^{-} \cup t_0^{-+}$ respectively) according to net effect considerations [26]. These relevant structural events are computed over relations in real-time by the active Algorithm 2.

Algorithm 2. Compute relevant instances that may modify DRs

```
   Input: A composite structural event $CSE$, $I$ and $D$ the auxiliary relations of
          base relation.
   Output: Auxiliary relations $I$ and $D$ updated for a $CSE$.
   Method:
1  foreach $PSE \in CSE$ do
2      if $(PSE = \Delta\ t^{-+})$ then                                /* update event */
3          if $(\{t_0^{-+} \cap I\} = \emptyset)$ then
4              insert into $I$ values $t_1^{-+}$;
5              insert into $D$ values $t_0^{-+}$;
6          else
7              update $u \in I$ set $u = t_1^{-+}$ where $u = t_0^{-+}$;
8      else if $(PSE = \Delta\ t^{+})$ then                            /* insert event */
9          insert into $I$ values $t^{+}$;
10     else                                           /* delete event ($PSE$ = $\Delta t^{-}$) */
11         if $(\{t^{-} \cap I\} = \emptyset)$ then
12             insert into $D$ values $t^{-}$;
13         else
14             delete $u \in I$ where $u = t^{-}$;
15 return $I, D$
```

This active process adds a minimum activity over regular data operations, just the necessary ones to store relevant instances and to apply net effect policies. The behavior of the algorithm is similar when considering only insertion and deletion events, but note the benefits of using update occurrences when a record is already inserted or modified. The rule base is updated only with these instances, by incrementing previous rules information. These rule base updates could be made automatically with a decision-maker's rule base access or scheduled. In this step, Algorithm 3 is presented in order to update *DRM*.

Algorithm 3. Deferred incremental maintenance for relevant instances

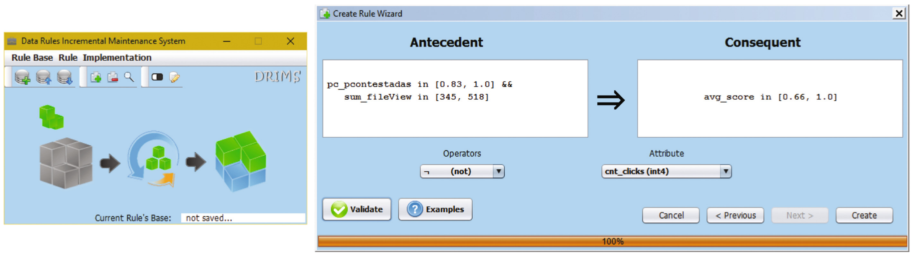
```
   Input: $I$, $D$ auxiliary relations of Algorithm 2 output, and measure-parts $DRM$.
   Output: An updated measure-parts $DRM$.
   Method:
1  foreach $Mp_k \in DRM$ do
2      update $Mp_k$, increment with $I$;
3      update $Mp_k$, decrement with $D$;
4  empty $I$;
5  empty $D$;
6  return $DRM$
```

This deferred proposal, like the immediate one, updates the rule base without access to the base relation. This is an important feature in a huge amount of data where the access to complete information is highly inefficient. Also, it entails the benefits of having only two auxiliary relations against more.

4 A Graphical User Interface

To increase the usability of our algorithms, we have designed DRIMS, a simple and friendly user interface Fig. 2 (left). It has a wizard-based interface, where through a few steps, users can create their own rules. The tool has two main possible applications: manage a repository of rules (such as create new rules, delete old rules or modify the existing ones) and the implementation of algorithms to maintain rules measures up-to-date in an efficient way.

The tool has three main menus, two of them are intended for rule's repository management and a third for the implementation of those rules into a business database. Creating rules in the repository is done through a multi-step wizard where the antecedent and consequent are defined as shown in Fig. 2 (right).



```
pc_pcontestadas in [0.83, 1.0] &&
    sum_fileView in [345, 518]
```

```
avg_score in [0.66, 1.0]
```

Fig. 2. The graphical user interface of DRIMS at start-up (left) and rule creation (right).

4.1 Prerequisites

As previously presented in the introduction, compilation and execution of DRIMS requires an installed and configured Java environment. The Java Runtime Environment (JRE) is a free software and may be obtained from ORACLE's web pages[1]. In the current version, DRIMS can implement algorithms in two of the main opensource database managements systems: PostgreSQL[2] and MySQL[3]. However, the tool also allows to create rules without maintaining a connection to the database.

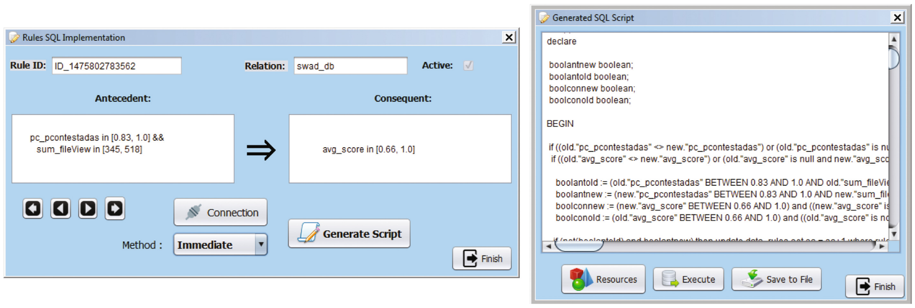
[1] Java Runtime Environment, https://www.java.com/en/download/.
[2] PostgreSQL Global Development Group, http://www.postgresql.org.
[3] MySQL Community Server, http://dev.mysql.com/downloads/mysql/.

4.2 Rules Implementation

The data rules are stored through an XML repository. The tool validates that repository using an XSD schema. For each rule, regardless of its type, the common attributes are stored. That includes the antecedent and consequent of the rule, the type to which it belongs and the relationship to which it refers. Note that in the case of FARs their linguistic labels are also stored for each existing attribute in the rule. In the case of ARs the tool processes quantitative rules' [31] as well as other more complex types of rules such as negative association rules [33].

Finally, the rules stored in the repository can be implemented directly in the business database as shown in Fig. 3 (left). The antecedent and consequent of the rules are grammatically parsed by a small translator made with ANTLR v4 [23] grammar parser. With the use of ANTLR/StringTemplates the SQL script that implements the incremental maintenance of the rule measure is generated.

The SQL script generated is composed primarily of active database resources [24] such as triggers. Besides the triggers, DRIMS tool can creates in business database catalog other objects such as views, functions and tables. These resources are created according to the chosen method and the rule's type to be implemented. In the graphical interface as shown by Fig. 3 (right) we can to list all resources of the generated script. This window also allows to run this script in the business database and save it as a plain text.


Rules SQL Implementation

Rule ID: ID_1475802783562
Relation: swad_db
Active: ✓

Antecedent:
pc_pcontestadas in [0.83, 1.0] &&
sum_fileView in [345, 518]

⇒

Consequent:
avg_score in [0.66, 1.0]

Method : Immediate

Generated SQL Script

```
declare

 boolantnew boolean;
 boolantold boolean;
 boolconnew boolean;
 boolconold boolean;

BEGIN

 if ((old."pc_pcontestadas" <> new."pc_pcontestadas") or (old."pc_pcontestadas" is nu
  if ((old."avg_score" <> new."avg_score") or (old."avg_score" is null and new."avg_sco

   boolantold := (old."pc_pcontestadas" BETWEEN 0.83 AND 1.0 AND old."sum_fileVie
   boolantnew := (new."pc_pcontestadas" BETWEEN 0.83 AND 1.0 AND new."sum_fil
   boolconnew := (new."avg_score" BETWEEN 0.66 AND 1.0) and ((new."avg_score" is
   boolconold := (old."avg_score" BETWEEN 0.66 AND 1.0) and ((old."avg_score" is no
```

Fig. 3. The graphical user interface for rule's implementation (left) and script's manipulation (right).

5 Experimental Results

The experiments over DRIMS algorithms have been designed to observe the different behaviors of the proposed methods in order to consider their implementation in real applications. The experiments also compare the proposed algorithms with those reported in the literature. These are being performed on real data and real structural events obtained from SWAD, a web system for education support at the University of Granada [5].

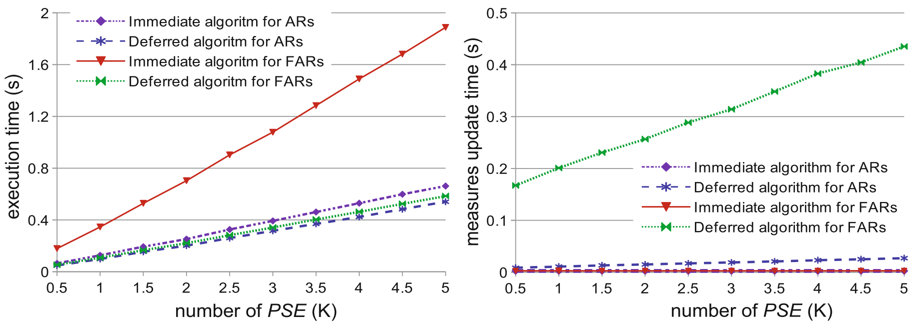


Fig. 4. Proposals comparison for ARs and FARs maintenance on execution times (left) and measures update times (right) in PostgreSQL.

Results illustrate the performance of proposed algorithms in order to maintain seven ARs, seven ADs and seven FARs. These rules were discovered using the KEEL data mining software tool [2]. Maintenance is implemented using the certainty factor metric from two open source database management systems: PostgreSQL Server version 9.2.2 and MySQL Server version 5.6.13. Both management systems show similar results. The experiments were carried out on a dedicated GNU/Linux server with eight processors i7-2600 at 3.4 GHz and 15 GB of main memory.

The experiments have been designed to observe two approaches' behavior: active process execution time and measures update time. The former presents consuming time when processing different numbers of primitive structural events on studied dataset. The latter exposes the consuming time for the rule measures update, after the same primitive structural events take place. The primitive structural events contain database insert, update, and delete operations extracted from real database transitions. In Fig. 4 these results are presented for ARs and FARs in PotsgreSQL.

The performance of proposed algorithms was also compared with traditional and incremental algorithms for FARs maintenance. In Fig. 5 a total execution time is presented for proposed algorithms, batch mining, and incremental mining methods for different datasets. These datasets were obtained from the UCI Machine Learning Repository [17]: the Diabetes 130-US hospitals for years 1999–2008 (diabetes), the Color Texture and the Color Moments parts of Corel Image Features. Details about these datasets can be found on the UCI Machine Learning website. For diabetes datasets nine attributes were selected. Seven FARs were extracted using KEEL data mining software tool from each dataset in order to be incrementally maintained by proposed algorithms.

Our proposal reflects the total time of executing 5 K data operations plus updates measures of rules in order to maintain the fuzzy rule base up-to-date. Batch and incremental mining methods reflect the mining execution time for the same goal. The fuzzy Apriori algorithm stands for batch mining methods. For incremental mining methods we only consider the fuzzy FP-growth [13] execution

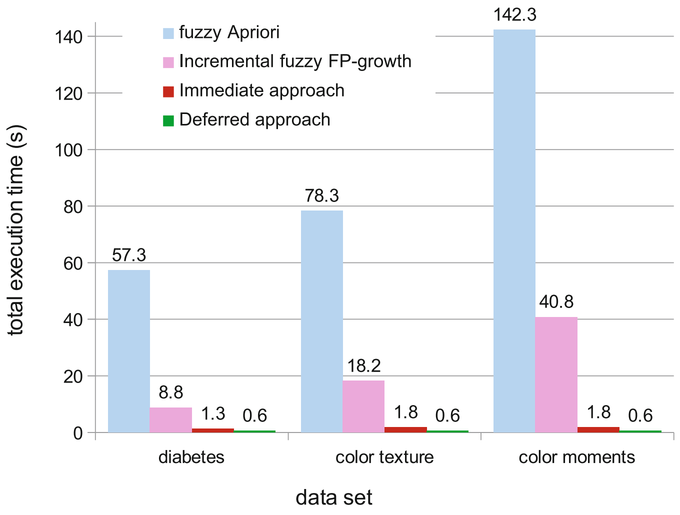


Fig. 5. Related and proposed algorithm comparison of total execution time for FARs maintenance in different datasets.

time and depreciate the fuzzy FP-tree build time, assuming it was incrementally maintained. This approach is referred to as incremental fuzzy FP-growth. For fuzzy Apriori and fuzzy FP-growth algorithms, three fuzzy regions were defined for numeric attributes. The minimum support threshold was set at 10% and minimum confidence threshold at 80%. Both mining algorithm experiments were developed using the KEEL data mining software tool.

6 Conclusion

In real-world applications, records are commonly inserted, updated or deleted outdating the previous extracted knowledge as inexact and invalid. In some scenarios, it is necessary to re-run traditional mining or incremental mining algorithms only for updating previous discovered rules. It is possible, from another perspective, to maintain the known rules incrementally by computing data changes efficiently.

In this paper, DRIMS is presented as the free and open-source software tool for incrementally maintaining previous discovered rules. It has a wizard-based interface, where through a few steps users can create and manage their own rules. This tool implements two algorithms for maintenance purpose of association rules, approximate dependencies and fuzzy association rules. Experimental results on real data and operations show that DRIMS maintenance proposals achieve a better performance against the batch mining and incremental mining approach. We believe this work represents a powerful enhancement to the incremental maintenance of previous discovered data rules and their implementation in business relational databases for real-time decision support.

Acknowledgements. This work has been partially supported by the Spanish Ministry of Economy and Competitiveness and the European Regional Development Fund - ERDF (Fondo Europeo de Desarrollo Regional - FEDER) under project TIN2014-58227-P *Descripción lingüística de información visual mediante técnicas de minería de datos y computación flexible*.

References

1. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. SIGMOD Rec. **22**(2), 207–216 (1993)
2. Alcalá-Fdez, J., Sánchez, L., García, S., Jesus, M., Ventura, S., Garrell, J., Otero, J., Romero, C., Bacardit, J., Rivas, V., Fernández, J., Herrera, F.: KEEL: a software tool to assess evolutionary algorithms for data mining problems. Soft Comput. **13**(3), 307–318 (2009)
3. Berzal, F., Blanco, I., Sánchez, D., Vila, M.A.: Measuring the accuracy and interest of association rules: a new framework. Intell. Data Anal. **6**(3), 221–235 (2002)
4. Berzal, F., Cubero, J.C., Marín, N., Serrano, J.M.: TBAR: an efficient method for association rule mining in relational databases. Data Knowl. Eng. **37**(1), 47–64 (2001)
5. Cañas, A., Calandria, D., Ortigosa, E., Ros, E., Díaz, A.: Swad: web system for education support. In: Fernández-Manjón, B., Sánchez-Pérez, J.M., Gómez-Pulido, J.A., Vega-Rodriguez, M.A., Bravo-Rodriguez, J. (eds.) Computers and Education: E-Learning, From Theory to Practice, pp. 133–142. Springer, Dordrecht (2007)
6. Cabot, J., Teniente, E.: Computing the relevant instances that may violate an OCL constraint. In: Pastor, O., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 48–62. Springer, Heidelberg (2005). doi:10.1007/11431855_5
7. Ceri, S., Widom, J.: Deriving production rules for incremental view maintenance. In: Proceedings of the 17th International Conference on Very Large Data Bases, pp. 577–589 (1991)
8. Cheung, D., Han, J., Ng, V., Wong, C.Y.: Maintenance of discovered association rules in large databases: an incremental updating technique. In: Proceedings of the Twelfth International Conference on Data Engineering, pp. 106–114 (1996)
9. Delgado, M., Marin, N., Sánchez, D., Vila, M.A.: Fuzzy association rules: general model and applications. IEEE Trans. Fuzzy Syst. **11**(2), 214–225 (2003)
10. Greco, S., Słowiński, R., Szczęch, I.: Properties of rule interestingness measures and alternative approaches to normalization of measures. Inf. Sci. **216**, 1–16 (2012)
11. Gupta, A., Mumick, I.S., et al.: Maintenance of materialized views: problems, techniques, and applications. IEEE Data Eng. Bull. **18**(2), 3–18 (1995)
12. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. SIGMOD Rec. **29**(2), 1–12 (2000)
13. Hong, T.P., Lin, T.C., Lin, T.C.: Mining complete fuzzy frequent itemsets by tree structures. In: 2010 IEEE International Conference on Systems Man and Cybernetics (SMC), pp. 563–567 (2010)
14. Lee, Y.S., Yen, S.J.: Incrementally mining frequent patterns from large database. In: Pedrycz, W., Chen, S.-M. (eds.) Information Granularity, Big Data, and Computational Intelligence. Studies in Big Data, vol. 8, pp. 121–140. Springer, Heidelberg (2015)
15. Lenca, P., Meyer, P., Vaillant, B., Lallich, S.: On selecting interestingness measures for association rules: user oriented description and multiple criteria decision aid. Eur. J. Oper. Res. **184**(2), 610–626 (2008)

16. Li, X., Deng, Z.-H., Tang, S.: A fast algorithm for maintenance of association rules in incremental databases. In: Li, X., Zaïane, O.R., Li, Z. (eds.) ADMA 2006. LNCS, vol. 4093, pp. 56–63. Springer, Heidelberg (2006). doi:10.1007/11811305_5
17. Lichman, M.: UCI machine learning repository (2013)
18. Lin, C.W., Hong, T.P.: Maintenance of pre large trees for data mining with modified records. Inform. Sci. **278**, 88–103 (2014)
19. Liu, C.-Y., Tseng, C.-Y., Chen, M.-S.: Incremental mining of significant URLs in real-time and large-scale social streams. In: Pei, J., Tseng, V.S., Cao, L., Motoda, H., Xu, G. (eds.) PAKDD 2013. LNCS, vol. 7819, pp. 473–484. Springer, Heidelberg (2013). doi:10.1007/978-3-642-37456-2_40
20. Liu, H., Lin, Y., Han, J.: Methods for mining frequent items in data streams: an overview. Knowl. Inf. Syst. **26**(1), 1–30 (2011)
21. Medina, R., Nourine, L.: A unified hierarchy for functional dependencies, conditional functional dependencies and association rules. In: Ferré, S., Rudolph, S. (eds.) ICFCA 2009. LNCS, vol. 5548, pp. 98–113. Springer, Heidelberg (2009). doi:10.1007/978-3-642-01815-2_9
22. Nakayama, H., Hoshino, A., Ito, C., Kanno, K.: Formalization and discovery of approximate conditional functional dependencies. In: Decker, H., Lhotská, L., Link, S., Basl, J., Tjoa, A.M. (eds.) DEXA 2013. LNCS, vol. 8055, pp. 118–128. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40285-2_12
23. Parr, T.: The Definitive ANTLR 4 Reference, 2nd edn. Pragmatic Bookshelf, Dallas (2013)
24. Paton, N.W., Díaz, O.: Active database systems. ACM Comput. Surv. **31**(1), 63–103 (1999)
25. Pérez-Alonso, A., Blanco, I., Serrano, J.M., González-González, L.M.: Drims: data rules incremental maintenance system (2016). https://github.com/AlainPerez/DRIMS-Repository
26. Pérez-Alonso, A., Medina, I.J.B., González-González, L.M., Serrano Chica, J.M.: Incremental maintenance of discovered association rules and approximate dependencies. Intell. Data Anal. **21**(1), 117–133 (2017)
27. Qin, S.J.: Process data analytics in the era of big data. AIChE J. **60**(9), 3092–3100 (2014)
28. Sánchez, D., Serrano, J.M., Blanco, I., Martín-Bautista, M.J., Vila, M.A.: Using association rules to mine for strong approximate dependencies. Data Min. Knowl. Disc. **16**(3), 313–348 (2008)
29. Sauter, V.: Decision Support Systems for Business Intelligence. Wiley, Hoboken (2014)
30. Shah, S., Chauhan, N., Bhanderi, S.: Incremental mining of association rules: a survey. Int. J. Comput. Sci. Inf. Technol. **3**(3), 4071–4074 (2012)
31. Srikant, R., Agrawal, R.: Mining quantitative association rules in large relational tables. ACM SIGMOD Rec. **25**, 1–12 (1996)
32. Tan, J., Bu, Y., Zhao, H.: Incremental maintenance of association rules over data streams. In: 2nd International Conference on Networking and Digital Society (ICNDS), vol. 2, pp. 444–447 (2010)
33. Wu, X., Zhang, C., Zhang, S.: Efficient mining of both positive and negative association rules. ACM Trans. Inf. Syst. **22**(3), 381–405 (2004)
34. Wu, X., Zhu, X., Wu, G.Q., Ding, W.: Data mining with big data. IEEE Trans. Knowl. Data Eng. **26**(1), 97–107 (2014)
35. Zia, Z.K., Tipu, S.K., Khan, M.I.: Research on association rule mining. Adv. Comput. Math. Appl. **2**(1), 226–236 (2012)