Data Mining in Databases: Languages and Indices

Elena Baralis, Tania Cerquitelli, Silvia Chiusano and Rosa Meo

Abstract Database systems methodologies and technology can provide a significant support to data mining processes. In this chapter we explore approaches which address the integration between data mining activities and DBMSs from different perspectives. More specifically, we focus on (i) specialized query languages which allow to define complex data mining tasks through the submission of query requests, and (ii) indices, i.e., physical data structures designed to improve the performance of mining algorithms.

Keywords Inductive databases · Data mining · Database indices · Association rules · Specialised query languages

1 Introduction

The topic of data mining is becoming every day more important since most any company and data center nowadays has accumulated in its history large volumes of data and is willing to analyse them and acquire knowledge in order to increase the competitive advantage over competitors or improve its business processes. The knowledge might be a model applicable to predict a target variable or descriptive, such that the user can use it in order to summarise the details of the data that cannot be analysed by a human given the large volumes of stored data. Of particular importance is the fact that very often the majority of these data, especially if it regards the past history of the business, such as clients information and past interaction, is stored in a

E. Baralis · T. Cerquitelli · S. Chiusano
Politecnico di Torino, corso Duca degli Abruzzi 24, Torino, Italy
e-mail: Elena.Baralis@polito.it

T. Cerquitelli
e-mail: Tania.Cerquitelli@polito.it

S. Chiusano
e-mail: Silvia.Chiusano@polito.it

R. Meo (✉)
Università di Torino, corso Svizzera 185, Torino, Italy
e-mail: Rosa.Meo@di.unito.it; meo@di.unito.it

persistent way in a DBMS which is almost always a relational database. An immediate conclusion arises: more technology and specialised systems should be developed by the database and data mining community in order to enhance and distribute facilities for the analysis of large volumes of data stored in relational databases.

Data mining is an interdisciplinary subfield of computer science aimed at discovering unexpected and potentially useful knowledge from large data collections. It includes a plethora of methods at the intersection of artificial intelligence, machine learning, statistics, and database systems, that can be profitably used to mine interesting patterns such as groups of similar data objects (cluster analysis), dependencies among data objects (association rule analysis), or a model describing data classes (classification). Traditionally data mining algorithms analyse large data collections stored into flat-files, even if possibly extracted from a DBMS. However this data extraction is not much practical, especially if a large volume of data is interested in the extraction. More effectively some research activity has addressed the application of data mining methods directly on relational data. Coupling data mining methods with the usual querying techniques can have a great potential in finally providing the end-user a richer, more diversified and interesting knowledge.

In this chapter we focus mainly on the knowledge that can be extracted from relational databases under the form of frequent patterns. In particular we focus on frequent itemsets and association rules that have been used with success in the past to solve predictive tasks such as classification [22] and to form a descriptive model of the dataset as well by the collection of frequent patterns extracted [3, 4].

Association rule mining, aiming at discovering correlations among data items, is the collection of data mining methods that can be more easily exploited for DBMS mining. Association rules are extracted from a transactional database $\mathcal{D}$. $\mathcal{D}$ is a collection of transactions, where each transaction is a set of data items. Association rules are usually represented in the form $A \rightarrow B$, where A and B are itemsets, i.e., sets of data items. Itemsets are characterized by their frequency of occurrence in $\mathcal{D}$, which is called support. Different constraints may be enforced to reduce the computational cost of itemset extraction, among which the most simple are support and item constraints [21, 37]. The support constraint enforces a threshold on the minimum support of the extracted itemsets. The item constraint enforces the extraction of the complete set of itemsets which include the required items.

Various efficient algorithms has been proposed for itemset extraction, which represents the most computationally intensive knowledge extraction task in association rule mining [2]. Ad-hoc main memory data structures are exploited to efficiently extract itemsets from data collections usually stored into binary files [2, 5, 17, 24, 28, 32, 34] Recently, disk-based extraction algorithms have been proposed to support the extraction from these large datasets [12, 15, 31].

In this chapter, we provide the following contributions: (i) an overview of the query languages and query optimisation techniques that can be used to interact with the database system and specify the type of patterns in which the analyst is interested in and (ii) a DBMS-based approach to support itemset mining queries. Relational DBMSs exploit indices, which are ad hoc data structures, to support the execution of complex queries. Following this approach, we describe the *IMine index*

(Itemset-Mine index), proposed in [6, 7], a compact data structure that provides a complete representation of transactional data supporting efficient itemset extraction from a relational DBMS.

2 An Overview of Specialised Query Languages for Data Mining

Inductive databases have been proposed to afford the problem of knowledge discovery from huge databases in [18]. This kind of databases integrates raw data with knowledge extracted from raw data, materialized under the form of patterns. With an inductive database the user/analyst performs a set of operations on data and on patterns using a specialized query language, powerful enough to perform all the required manipulations to support the KDD process, such as data preprocessing, pattern discovery and pattern post-processing.

- Selection of data to be mined.
- Specification of the type of patterns to be mined (descriptive or predictive).
- Specification of the background knowledge, for instance under the form of a concept hierarchy or ontology.
- Definition of constraints that the extracted patterns must satisfy in order to allow the user to specify the interesting patterns. This occurs usually by using measures like frequency, generality, coverage, similarity, novelty, etc.
- Satisfaction of the closure property (by storing the results in the database).
- Post-processing of results in order to allow the user to interact with the extracted patterns by browsing, apply selection templates, cross over patterns and data by selection of the data in which some patterns hold, or aggregating results.

A few query languages can be considered as candidates for inductive databases. We present a brief comparison between query languages that have been proposed for association rules extraction: MSQL, DMQL and MINE RULE. This allows us to compare the language design guidelines, with particular attention to the features of inductive databases for which they are designed.

2.1 MSQL

MSQL has been described in [19]. It comprises the following statements:

GetRules: it generates rules into a rule base;
SelectRules: it queries the rule base;
Create Encoding: it efficiently encodes discrete values into continuous valued attributes;

Satisfies and violates: they allow to cross-over data and rules, and that can be used in a data selection statement.

The main features of this language are the following. It has the ability to nest SQL expressions such as sorting and grouping in a unique MSQL statement. It satisfies the closure property and provides operators to manipulate results of previous queries. It can perform a cross-over between data and rules with operations allowing to identify subsets of data satisfying or violating a given set of rules. It distinguishes between rule generation and rule querying. Indeed, as the volume of generated rules might explode, rules might be extensively generated only at querying time, and not at generation time.

2.2 DMQL

DMQL has been presented in [16]. It consists of the specification of four major primitives for the management of:

1. the set of relevant data w.r.t. a data mining process; this is specified by means of conventional query.
2. the kind of knowledge to be discovered; it includes association rules, classification rules, characteristic descriptions that are a summarization of the common properties of the data, comparisons descriptions that discriminate the tuples belonging to a class with different classes, generalized relations obtained by generalizing a set of data according to the conceptual level described in a specified concept hierarchy.
3. the background knowledge by providing a set of primitives for the management of a set of concept hierarchies or generalization operators that assist the generalization processes.
4. the justification of the interestingness of the knowledge (i.e., by evaluation measure thresholds such as association rules measures like the classical support and confidence thresholds, the allowed noise and rule novelty).

2.3 MINE RULE

MINE RULE has been originally presented in [26]. This operator extracts a set of association rules from the database and stores them back in the database in a separate relation. This language is an extension of SQL. Its main features are the following.

1. Selection of the relevant set of data for a data mining process; this selection is applied at different granularity levels, that is at the row level (selection of a subset of the rows of a relation) or at the group level (group condition).

2. Definition of the structure of the rules to be mined and of constraints applied at different granularity levels; it might define either unidimensional association rule (i.e., rules elements are values of the same dimension or attribute), or multidimensional. Furthermore, rules constraints can be applied at the rule level (mining conditions) in order to filter single rules, or can be applied as cluster conditions, in order to build separately the two parts of the rules (body or head).
3. Definition of the grouping condition that determines which data of the relation can take part to an association rule;
4. Definition of rule evaluation measures (i.e., support and confidence thresholds).

2.4 Optimization of Mining Queries

In [25] we highlighted the relationships between two mining queries: equivalence, inclusion and dependence.

Equivalence: Two queries are equivalent if for all instances of the source data each rule r in the result set of the first query is also in the result set of the second query and vice versa with the same value of rules evaluation measures (support and confidence).

Inclusion: A first query is included in the second one if for all instances of the source data each rule r in the result set of the first query is also in the result set of the second query with the same value of the rules evaluation measures.

Dominance: A first query is dominated by a second one if for all instances of the source data each rule r in the result set of the first query is also in the result set of the second query. Furthermore, in the result of the second query the values of the rules evaluation measures are an upper bound of the values of the corresponding rules from the second query.

We showed the practical implications of the discussed principles with a set of algorithms designed for MINE RULE. These algorithms use also a new designed mining index called mining that allows to reduce the portion of database to be read in response to some classes of queries. In these cases the workload of the mining engine is greatly reduced or completely saved.

In [27] we proposed to optimize constraint-based queries on itemsets with the aim to reduce the overall computation time of a mining query. We introduced a very generic constraint-based language for the extraction of frequent itemsets and presented an optimization scheme that exploits the available materialization of previous queries. The optimization scheme proposed is based on query rewriting. We studied the conditions under which query rewriting is possible and suggested a way to find such a rewriting. For efficiency, we proposed a composition scheme of the materializations that makes usage of common and efficient operations in DBMSs, i.e., intersection and union of relations.

3 Using Indices to Mine Frequent Patterns

A cornerstone of efficient query processing in relational DBMSs is the exploitation of indices. An index is a specialized data structure that supports selective access to the subset of physical pages needed to process a query. A variety of different data structures have been proposed to support data access both for the relational and non-relational data representations.

While several disk-based data structures have been proposed to support itemset mining (e.g., [9, 12, 15, 34] which are further discussed in Sect. 3.3), the definition of index structures to support frequent pattern mining in relational DBMSs has been addressed only in [6, 7]. The *IMine index* (Itemset-Mine index) [6, 7] has been fully integrated into the PostgreSQL DBMS kernel [30]. It is a persistent data structure that provides efficient and effective data access to itemset mining algorithms. More specifically, the IMine index provides a compact and complete representation of transactional data and is characterized by several important properties.

Complete representation. The IMine index is created without enforcing any constraint (e.g., support or item constraint). Hence, it is a *covering index*, i.e., itemset mining can be performed by means of the index alone, without accessing the original database. The IMine index provides a *complete* representation of the transactional data, thus supporting itemset extraction with arbitrary support thresholds.

General structure. The structure of the IMine index is designed to support a variety of itemset extraction algorithms. These algorithms are typically characterized by different in-memory data representations (e.g., array list, prefix-tree) and techniques to explore the search space. The IMine index features efficient data access methods to load in memory the data needed by the considered extraction algorithm. The enforcement of item constraints is also supported by the IMine index access methods. Finally, the generality of the index structure allows it to gracefully adapt to both sparse and dense data distributions.

Efficient data access. The physical organization of the IMine index is designed to provide efficient access to physical data blocks during the mining process. Correlated data are stored in the same physical block, thus allowing a significant reduction of the number of block reads.

The IMine index exploits PostgreSQL open source DBMS [30] physical level access methods. The performance of the approach has been compared with state-of-the-art algorithms (i.e., Prefix-Tree [14] and LCM v.2 [35]) accessing binary data on a flat file. The IMine index always provides a better performance, which also scales linearly for large datasets.

3.1 IMine Index Structure

The IMine index is characterized by two levels of indexing. The first level stores the transactional data in a compact prefix-tree structure which provides a lossless representation of the data, the *Itemset-Tree* (*I-Tree*). The second level allows reading selected I-Tree portions during itemset mining. It is a B+Tree structure, the *Item-Btree* (*I-Btree*), that stores the physical location of all item occurrences in the I-Tree. Hence, it provides efficient access to the I-Tree data blocks to load in memory the transactions including a selected item.

IMine data access methods. Different itemset mining algorithms may exploit the IMine index structure to load data in memory. Three different data access methods are available, each one providing an in-memory representation appropriate for the selected mining algorithm (e.g., FP-tree for FP-growth [17], array-based structure for LCM [35]). These methods access different parts of the IMine index, depending on the adopted itemset mining algorithm and on the enforced support and/or item constraints. More specifically, the following data access methods have been designed. The *Frequent-item based projection,* method supports projection-based algorithms (e.g., FP-growth [17]), while the *Support-based projection,* supports level-based (e.g., APRIORI [2]), and array-based (e.g., LCM v.2 [35]) algorithms. Finally, the *Item-based projection,* loads in memory all transactions including a given item and is exploited for item constrained mining.

IMine is a covering index. Hence, the original transactional database is not accessed. During the mining process, only a small portion of the entire dataset is actually loaded in memory for the local search performed by the extraction algorithm. By accessing the IMine index, only the relevant index blocks are loaded in memory, thus significantly reducing the number of disk reads. Read disk blocks are stored in the buffer cache memory of PostgreSQL. Furthermore, a very limited data portion is actually loaded in memory at each step of the algorithm. Hence, more memory space becomes available for the mining process.

IMine physical organization. The design of the physical organization of the IMine index aims at minimizing the cost of reading the data needed by the current step of the mining algorithm. The selection of the blocks including the paths of interest is performed by means of the I-Btree. Thus, the number of disk blocks read to load the required I-Tree paths is the most important factor contributing to the I/O cost. The correlation between index parts, i.e., data paths accessed together during the current mining step, is exploited to reduce the I/O cost. More specifically, correlated index parts are stored together in the same disk block.

Furthermore, the I-Tree is partitioned in three distinct layers. The intuition driving the partitioning process is that items with very low support do not satisfy most support constraints. Hence, they are accessed only rarely during the mining process. The nodes corresponding to low-support items belong to the lower levels of the I-Tree. More specifically, the frequency of the node accesses performed by the mining process is considered to perform the partitioning. The interaction of the following three factors affects node access frequency: (a) the node support, which shows the number of paths including the node, (b) the global support of the item associated to the node, and (c) the distance of the considered node from the root, described by the node level in the tree.

3.2 Itemset Mining

The IMine index can support a variety of different itemset mining algorithms. The main difference among different approaches is in (a) the main memory data structure exploited to store the required data, and (b) the strategy adopted by the algorithm to visit the search space. IMine data access methods load in memory the data needed by the current step of the itemset mining algorithm. In each step of the mining process, data is read from the I-Tree and loaded in memory, in the appropriate structure for the selected mining algorithm. Next, mining takes place on the loaded data.

Enforcing constraints. The specification of constraints on the extraction process yields a subset of (more) interesting itemsets and may help the human analyst in focusing on relevant knowledge. Pushing constraint enforcement into the mining process would allow early pruning the search space, thus improving the efficiency of the mining process. Hence, a significant research activity has been focused on the definition of strategies for constrained itemset extraction [10, 11, 21, 29, 33]. A classification of constraints into anti-monotonic, monotonic, succinct, and convertible has been proposed in [29], which also addresses constraint enforcement into the FP-growth algorithm. The IMine index can directly support the access strategies described in [29]. More specifically, the items of interest can be straightforwardly selected by accessing the I-Btree.

3.3 Disk-Based Strategies to Support Frequent Pattern Mining

Several approaches have been proposed to support itemset extraction from flat file by means of disk-based data structures. These approaches do not support the tight integration of itemset extraction in a relational DBMS. However, they exploit some form of file indexing structure to support the mining process.

An interesting hybrid in-core/out-of-core approach has been presented by Lucchese et al. [23]. Even though the disk is exploited as an auxiliary means to extend scalability, the mining process is still mainly memory-based [23]. An index structure based on signature files is proposed in [20]. It supports the candidate frequent itemsets generation process, but the actual candidate frequency check requires further dataset access.

Fully disk-based mining algorithms have been proposed to support the extraction of knowledge from large datasets (e.g., B+tree-based indices [34], Inverted Matrix [12], Diskmine [15] I/O conscious optimizations [9], DRFP-tree [1], VLDB-Mine [8]). The Inverted Matrix is a disk-based data structure proposed by El-Hajj and Zaïane [12] to store the transactional dataset in an inverted matrix layout. The proposed data structure deals well with very sparse datasets, in which a large number of items are characterized by unitary support. B+tree-based indices to access data have been proposed by Ramesh et al. [34]. The adopted data representation is either vertical (e.g., ECLAT-Based [36]) or horizontal (e.g., APRIORI-Based [2]).

In [15] large databases are materialized on disk by storing different (recursively generated) projected databases whose size fits main memory. Each projection, represented as an FP-tree, is first materialized on disk, and then separately loaded in main memory for itemset extraction. Diskmine allows efficient memory saving and maximizes memory exploitation. However, storing all projections may require significant disk space, and subsequently cause a non-negligible I/O cost during the mining process.

The path tiling approach, proposed by Buehrer et al. [9, 13], is an efficient itemset mining techniques exploiting I/O conscious optimizations. More specifically, several data locality strategies are proposed to reduce the number of reads during the mining process. However, different data structures and mining algorithms may have different data locality requirements. Hence, different I/O conscious techniques should be defined for different mining approaches.

In [8] a persistent and hybrid structure, named VLDBMine, is proposed to compactly store huge transactional datasets characterized by a variable data distribution. VLDBMine has been designed to support existing in-core algorithms by enhancing memory usage, thus achieving scalability through different selective data retrieval methods.

4 Conclusions

The full integration of data mining techniques as DBMS services is a challenging goal yet to be achieved. In this chapter, we presented different integration attempts, which address both the query/mining specification language and the definition of physical data structures to improve the performance of mining algorithms. From one side, different SQL-like mining languages have been proposed to ease the continuous and exploratory interaction between the user and the DBMS. From the other side, the IMine index, discussed in this chapter, supports efficient itemset mining into a relational DBMS. It has been implemented into the PostgreSQL open source DBMS and it profitably exploits its services, among which physical level access methods and DBMS buffer management.

In the future we hope that new specialised query languages and index structures will be proposed in the same vein to facilitate the interaction of the user and the data analysis on the very big data stored in the so-called No-SQL databases.

References

1. M. Adnan, R. Alhajj, Drfp-tree: disk-resident frequent pattern tree. Appl. Intell. Springer **30**(2), 84–97 (2009)
2. R. Agrawal, R. Srikant, Fast algorithms for mining association rules in large databases, in *VLDB '94* (1994), pp. 487–499

3. R. Agrawal, R. Srikant, Mining sequential patterns, in *International Conference on Data Engineering*, Taipei, Taiwan, March 1995
4. R. Agrawal, T. Imielinski, A. Swami, Mining association rules between sets of items in large databases, in *Proc.ACM SIGMOD Conference on Management of Data*(British Columbia, Washington, D.C., 1993), pp. 207–216
5. R. Agrawal, T. Imilienski, A. Swami, Mining association rules between sets of items in large databases, in *SIGMOD'93*, Washington DC, May 1993
6. E. Baralis, T. Cerquitelli, S. Chiusano, Index support for frequent itemset mining in a relational dbms, in *ICDE* (2005), pp. 754–765
7. E. Baralis, T. Cerquitelli, S. Chiusano, Imine: index support for item set mining. IEEE Trans. Knowl. Data Eng. **21**(4), 493–506 (2009)
8. E. Baralis, T. Cerquitelli, S. Chiusano, A. Grand, Scalable out-of-core itemset mining. Inf. Sci. **293**, 146–162 (2015)
9. G. Buehrer, S. Parthasarathy, A. Ghoting, Out-of-core frequent pattern mining on a commodity pc, in *KDD '06* (2006), pp. 86–95
10. Y.-L. Cheung, Mining frequent itemsets without support threshold: with and without item constraints. IEEE Trans. Knowl. Data Eng. **16**(9), 1052–1069 (2004). Member-Ada Wai-Chee Fu
11. G. Cong, B. Liu, Speed-up iterative frequent itemset mining with constraint changes, in *ICDM* (2002), pp. 107–114
12. M. El-Hajj, O.R. Zaiane, Inverted matrix: Efficient discovery of frequent items in large datasets in the context of interactive mining. in *ACM SIGKDD* (2003)
13. A. Ghoting, G. Buehrer, S. Parthasarathy, D. Kim, A. Nguyen, Y.-K. Chen, P. Dubey, Cache-conscious frequent pattern mining on modern and emerging processors. VLDB J. **16**(1), 77–96 (2007)
14. G. Grahne, J. Zhu, Efficiently using prefix-trees in mining frequent itemsets, in *FIMI*, November 2003
15. G. Grahne, J. Zhu, Mining frequent itemsets from secondary memory, in *ICDM '04* (IEEE Computer Society, Washington, DC, USA, 2004), pp. 91–98
16. J. Han, Y. Fu, W. Wang, K. Koperski, O. Zaiane, DMQL: a data mining query language for relational databases, in *Proceedings of SIGMOD-96 Workshop on Research Issues on Data Mining and Knowledge Discovery* (1996)
17. J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, in *SIGMOD '00* (2000), pp. 1–12
18. T. Imielinski, H. Mannila, A database perspective on knowledge discovery. Commun. ACM **39**(11), 58–64 (1996)
19. T. Imieliński, A. Virmani, Msql: a query language for database mining. Data Min. Knowl. Disc. **3**(4), 373–408 (1999)
20. B. Lan, B.C. Ooi, K.-L. Tan, Efficient indexing structures for mining frequently patterns, in *IEEE ICDE* (2002)
21. C.K.-S. Leung, L.V.S. Lakshmanan, R.T. Ng, Exploiting succinct constraints using fp-trees. SIGKDD Explor. Newsl. **4**(1), 40–49 (2002)
22. B. Liu, W. Hsu, Y. Ma, Integrating classification and association rule mining, in *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining, KDD'98* (AAAI Press, 1998), pp. 80–86
23. C. Lucchese, S. Orlando, R. Perego, kdci: on using direct count up to the third iteration, in *FIMI* (2004)
24. H. Mannila, H. Toivonen, A. Inkeri Verkamo, Efficient algorithms for discovering association rules, in *KDD Workshop* (1994), pp. 181–192
25. R. Meo, Optimization of a language for data mining, in *Proceedings of the 2003 ACM Symposium on Applied Computing*, Melbourne, Florida, 2003
26. R. Meo, G. Psaila, S. Ceri, A new SQL-like operator for mining association rules, in *Proceedings of the 22st VLDB Conference*, Bombay, India, September 1996

27. R. Meo, M. Botta, R. Esposito, *Query Rewriting in Itemset Mining* (Springer, Berlin, 2004), pp. 111–124
28. S. Orlando, C. Lucchese, P. Palmerini, R. Perego, F. Silvestri, kDCI: a multi-strategy algorithm for mining frequent sets, in *FIMI* (2003)
29. J. Pei, J. Han, L.V.S. Lakshmanan, Pushing convertible constraints in frequent itemset mining. Data Min. Knowl. Discov. **8**(3), 227–252 (2004)
30. PostgreSQL. Postgresql, http://www.postgresql.org
31. G. Ramesh, W.A. Maniatty, M.J. Zaki, Indexing and data access methods for database mining, in *DMKD* (2002)
32. A. Savasere, E. Omiecinski, S.B. Navathe, An efficient algorithm for mining association rules in large databases, in *VLDB* (1995), pp. 432–444
33. R. Srikant, Q. Vu, R. Agrawal, Mining association rules with item constraints, in *KDD* (1997), pp. 67–73
34. H. Toivonen, Sampling large databases for association rules, in *VLDB* (1996), pp. 134–145
35. T. Uno, M. Kiyomi, H. Arimura, LCM ver. 2: efficient mining algorithms for frequent/closed/maximal itemsets, in *FIMI '04* (2004)
36. M.J. Zaki, Scalable algorithms for association mining. IEEE Trans. Knowl. Data Eng. **12**(3), 372–390 (2000)
37. L. Zhao, M.J. Zaki, N. Ramakrishnan, BLOSOM: a framework for mining arbitrary boolean expressions, in *KDD '06* (ACM Press, New York, NY, USA, 2006), pp. 827–832