SQL-Based KDD with Infobright's RDBMS: Attributes, Reducts, Trees*

Jakub Wróblewski[1] and Sebastian Stawicki[2]

[1] Infobright Inc.
ul. Krzywickiego 34, lok. 219, 02-078 Warsaw, Poland
[2] Institute of Mathematics, University of Warsaw
ul. Banacha 2, 02-097 Warsaw, Poland
jakubw@infobright.com, stawicki@mimuw.edu.pl

Abstract. We present a framework for KDD process implemented using SQL procedures, consisting of constructing new attributes, finding rough set-based reducts and inducing decision trees. We focus particularly on attribute reduction, which is important especially for high-dimensional data sets. The main technical contribution of this paper is a complete framework for calculating short reducts using SQL queries on data stored in a relational form, without a need of any external tools generating or modifying their syntax. A case study of large real-world data is presented. The paper also recalls some other examples of SQL-based data mining implementations. The experimental results are based on the usage of Infobright's analytic RDBMS, whose performance characteristics perfectly fit the requirements of presented algorithms.

Keywords: KDD, Rough sets, Reducts, Decision trees, Feature extraction, SQL, High-dimensional data.

1 Introduction

The process of knowledge discovery in databases consists of many stages, such as data integration and cleaning, feature extraction, dimension reduction, model construction. With growing sizes of data, there are a number of approaches attempting to mine the available sources (often in a form of multitable relational databases) by means of database query languages rather than operating directly on files. Such approaches utilize additional tables containing some precalculated or partial results used in implemented algorithms' runs; the other trend is to split the main task into subtasks solved by relatively basic SQL queries. Our work concentrates on implementation of known KDD tools (feature extraction, decision reducts, decision trees) directly in SQL-based environment.

* The second author was partly supported by Polish National Science Centre (NCN) grants DEC-2011/01/B/ST6/03867 and DEC-2012/05/B/ST6/03215, and by National Centre for Research and Development (NCBiR) grant SP/I/1/77065/10 by the strategic scientific research and experimental development program: "Interdisciplinary System for Interactive Scientific and Scientific-Technical Information".

M. Kryszkiewicz et al. (Eds.): RSEISP 2014, LNAI 8537, pp. 28–41, 2014.

Most of data mining tools are based on a paradigm of one decision table. On the other hand, large data sets often have a form of relational, multi-table databases. In these cases a data mining process is preceded by preprocessing steps (based on domain-dependent knowledge), usually done interactively by a user. After these steps one joint data table is created. In [28] we presented an algorithm for automatic induction of such table, employing genetic algorithms and SQL-based attribute quality measures.

The next step in KDD process is an attribute subset selection [12,13]. It provides the basis for efficient classification, prediction and approximation models. It also provides the users with an insight into data dependencies. In this paper, we concentrate on attribute subset selection methods originating from the theory of rough sets [16]. There are numerous rough set-based algorithms aimed at searching for so called decision reducts – irreducible subsets of attributes, which (according to one of its most popular formulations) enable to discern sufficient amount of pairs of objects belonging to different decision classes. For a number of purposes, it is also useful to search for multiple decision reducts, containing possibly least overlapping sets of attributes [25]. This way, the users can observe multiple subsets of attributes providing roughly the same or complementary information. This may be important, e.g., for a design of ensembles of classifiers learnt over particular decision reducts, as combining classifiers is efficient especially if they are substantially different from each other [11].

There are numerous approaches to search for the shortest (or optimal with respect to various evaluation measures) reducts in large data [3]. One can combine basic top-down and bottom-up heuristics controlling a flow of adding and removing attributes with some elements of randomization, parallelism and evolution [17]. One can also use some modern computational paradigms, such as MapReduce [4], in order to scale computations with respect to the number of objects [30]. Finally, when analyzing texts [6], signals [24], images [26], microarrays [27] etc., some techniques aimed at attribute sampling and attribute clustering can help us to scale with respect to the number of dimensions [7].

In this paper, we follow another way of scaling decision reduct computations, namely, utilizing SQL-based scripts running against analytic databases. The idea is not new [5,10]. However, it seems that so far it did not get enough attention with respect to such aspects as: 1) easiness of operating with SQL while implementing initial designs of new attribute reduction approaches, 2) importance of choosing appropriate database systems and an efficient form of storing data information, and 3) performance benefits while searching for multiple reducts. Our methodology bases on EAV (Entity-Attribute-Value) data representation, which is suitable for storing sparse, multidimensional data in RDBMS.

The next step is an induction of predictive model in a form of decision rules or a decision tree. The idea of decision tree induction using SQL is not new [14]. In [10] the tree induction process is rewritten in a form of queries executed against EAV. In each induction step a new level of tree is added and the approach is based on a rule of choosing the most promising attribute from data. The authors introduce a measure which evaluates attributes and expresses their usefulness

according to a simple heuristic approach. A measure value is computed using a single SQL statement that evaluates usefulness of attributes (all at once) based on analysis of diversity of their values' averages in particular decision classes. The major advantage of the heuristic is the speed of execution, especially for the RDBMS solutions designed for such types of aggregations. Then using the method a cut on its range is created in order to possibly maximally separate objects from different decision classes. One of the main assets of using EAV model is used here: it enables to significantly limit number of queries by assigning the cuts for all nodes at a given level with a single SQL query.

The paper is organized as follows: Section 2 describes Infobright's database system – the data storage engine of our choice. Section 3 outlines the task of feature extraction. Section 4 discusses an EAV format with a special case of so called discernibility tables. Section 5 presents queries realizing some atomic rough set operations for EAV representation. Section 6 introduces some SQL-based attribute reduction scripts. Section 7 refers to the task of decision tree induction. Section 8 outlines our experimental framework and results obtained for the task of attribute reduction. Section 9 concludes the paper.

2 RDBMS Engine

In our research, we used the Infobright's database system, whose algorithms accelerate performance of SQL statements by an internal usage of rough set principles of computing with granulated data [21,22]. It is also important to mention about Infobright's approximate query extensions [8,19]. Although those extensions were not yet utilized in our attribute reduction procedures, they can be useful for bigger data sets in future. As for our current framework, the following aspects of Infobright are the most significant:

- Data compression [1,9]: Storing and processing large amount of data is feasible. For example, for the AAIA'2014 data set (see section 8.1), a discernibility table in EAV format of raw size 40 GB was compressed to 1.7 GB after loading into the Infobright's data table.
- SQL performance: Infobright's layer of rough information about the contents of granulated data is automatically employed by internal mechanisms to minimize the data access intensity while executing queries. These mechanisms turn out to be especially efficient for complex analytic types of SQL statements, such as those reported in this paper.
- Ease of usage: Unlike some other SQL and NoSQL data processing platforms, Infobright does not require any kind of sophisticated tuning. In particular, all existing parallel and caching mechanisms are ready to be used automatically. It helped a lot in an iterative design of our SQL scripts.

3 Construction of New Attributes

In many real-life KDD tasks the input data is given as a multitable relational database rather than a single decision table. In general we can assume, that any

descriptive or predictive problem in KDD [12] is concerned with a selected (or added) decision attribute in one of existing or newly created data tables (i.e. concerns one entity in relational database). If so, we have a distinguished table, which can be analyzed using standard data mining tools. On the other hand, only a part of information available in the database is collected in this table. Most of knowledge is distributed over relations and other tables.

In [28] we proposed a framework to mine dimension tables in a star (or snowflake) schema. The framework consists of a set of possible transitions (joins, aggregations), defined based on domain-dependent expert knowledge as well as on the interpretation of relations in database, and also quality measures to evaluate potential new attributes. Such attributes may be evaluated without actual materialization, basing on definitions expressed in SQL.

The main problem one will face using this scheme is a huge number of possible transitions (new attributes). However, since the process of attributes' quality calculation is faster than creation of classification algorithm, one can try to find good examples of attributes in reasonable time. An adaptive process of discovering new features based on genetic algorithms was proposed and implemented in [28]. To formalize this approach basing on rough set theory, a new notion of relational information system was introduced. An adaptive classification system based on predictive attributes' subsets quality measure was successfully used to analyze a real-life database. It is worth noting that analysis needs only a little preprocessing by a user; there is no need to create joint tables etc. since all operations are performed in SQL on original data. Only the final decision table consisting all derived attributes should be materialized in either classic or EAV form to perform further steps of data mining.

4 EAV and Discernibility Tables

In a case of Entity-Attribute-Value (EAV) form of data storage, a decision table is stored (e.g. in a relational database) as a collection of tuples (e, a, v), where e is an identifier of object (e.g. a row number in a classical decision table), a is an identifier of attribute, and v is an attribute value. EAV table is a convenient form for some cases of data mining/KDD tasks. In our algorithms, the main advantages of EAV are as follows:

- It is easily expandable, both in terms of entities (rows in a classical version of decision table) and attributes (columns). Adding new attributes does not require changing database schema nor reloading data. In particular, it may be useful to flexibly write down the results of the SQL-based feature extraction procedures referred in the previous section.
- Although it requires more space in general (three numbers for row/column, comparing to one in a classical case), but for sparse data containing a lot of NULL (unknown) values or for imbalanced attributes (when only positive information is actually stored), this form may occupy much less space. This makes various extensions of EAV format quite convenient in projects related to, e.g., scientific document indexing [20].

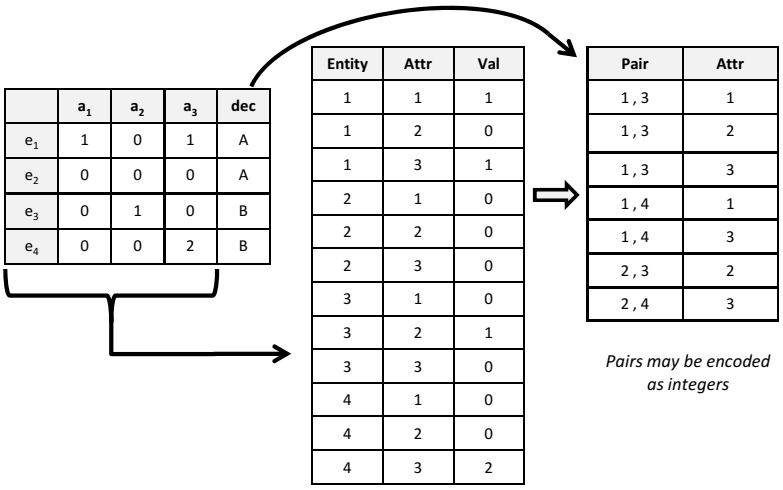

Fig. 1. Preparing a discernibility table (right) based on decision attribute of the original data table (left) and attribute values stored in EAV form (middle)

– Attributes are identified as numbers, and may be processed by SQL, allowing e.g. to operate on subsets of them, constructed dynamically or stored in tables. There is no need to construct a modified SQL query by external tools when another subset of attributes is analyzed. This makes EAV quite popular in various machine learning implementations, e.g., various extensions of association/decision rule mining procedures [18].

The above advantages has led also to EAV-based implementations of some rough set methods [10,23]. As for the task of calculation of decision reducts, we actually do not need to use an original decision table content. It is sufficient to store so called discernibility table in an EAV-like form, containing an identifier of an object pair and a number of the attribute:

```
CREATE TABLE discern (pair BIGINT, a INT);
```

For an input decision table of the form $\mathbf{A} = (U, A \cup \{dec\})$ [16], the content of discernibility EAV table is defined as follows:

$$\forall_{e_1,e_2 \in U; a \in A}\ a(e_1) \neq a(e_2) \wedge dec(e_1) \neq dec(e_2) \Leftrightarrow (id(e_1, e_2), a) \in discern \quad (1)$$

where dec is a distinguished decision attribute and $id(e_1, e_2) = e_1 + |U|e_2$ is a function encoding a pair of row numbers into one unique identifier. See Figure 1 for an outline of the whole process. Note that such created data table straightforwardly corresponds to the content of a discernibility matrix for $\mathbf{A}$ and, therefore, enables to implement analogous operations of attribute reduction [3].

In many cases the discernibility table is too large to be explicitly stored. As its size is $O(|U|^2|A|)$, data sets with too many objects must be analyzed by methods

avoiding explicit generation of discernibility structures. On the other hand, even high-dimensional data (with large $|A|$) with moderate number of objects ($|U|$ up to an order of tens of thousands) is still easy to store in an ordinary RDBMS. Additionally, if attribute values are sparse (majority of 0 or NULL values) and the decision classes are imbalanced, then the actual number of pairs of objects to be discerned, i.e., the rows in the EAV form of the discernibility table may be far less than the upper bound.

5 Rough Set Notions in SQL

Many of rough set related tasks may be easily done by SQL queries operating on the EAV discernibility tables. Below we present some examples.

5.1 Superreducts

Let $@all\_pairs$ be a variable storing a number of all pairs to be discerned:

```
SET @all_pairs = SELECT COUNT(DISTINCT pair) FROM discern;
```

Then the following query will check whether a subset of attributes $a_1, a_2, ...a_k \in A$ is sufficient to discern all pairs of objects:

```
SELECT COUNT(DISTINCT pair) FROM discern WHERE a IN (a1,...,ak);
```

If the answer is equal to $@all\_pairs$, then the subset is a superreduct.

As already mentioned, some SQL-based reduct calculation techniques are already known [5]. However, those methods usually require dynamic construction of queries by an external tool (e.g. a Java or C++ program). In our case, analogously to other EAV-based approaches [10,23], all queries are of the same form regardless of the considered subsets (or families of subsets) of attributes. A subset to be checked can be stored in a separate table and provided in form of a subquery or join. Below we denote such table as *candidates*:

```
SELECT COUNT(DISTINCT pair) FROM discern
  WHERE a IN (SELECT a FROM candidates);
```

This observation could make the process of data analysis much simpler for SQL-based users. The whole process of attribute reduction can be done automatically or semi-automatically by a set of queries and SQL procedures.

5.2 Decision Reducts and Approximate Reducts

To make sure a superreduct is actually a reduct, we should try to exclude all attributes one by one and make sure that the remaining subset is not sufficient. It can be done by single query, where $t2.a$ is an attribute to be excluded:

```
SELECT t2.a AS a_excl, COUNT(DISTINCT pair) AS cnt
  FROM discern AS t1 JOIN candidates AS t2
  WHERE t1.a IN (SELECT a FROM candidates) AND t1.a <> t2.a
GROUP BY a_excl HAVING cnt = @all_pairs;
```

If the answer is empty, then the *candidates* set is a reduct. Similar procedure may be applied to check whether a subset is an approximate (super)reduct [15]. It is sufficient to replace $@all\_pairs$ in the above queries with $@nearly\_all\_pairs = @all\_pairs \cdot (1 - \varepsilon)$ for a given error rate ε.

5.3 Pairwise Cores

Unlike classical cores which do not occur in data sets so often, pairwise cores are pairs of attributes such that each reduct needs to include at least one of them [29]. Such pairs can be easily calculated as follows:

```
SELECT t1.pair, MIN(a), MAX(a) FROM discern AS t1,
 (SELECT pair, COUNT(*) AS cnt FROM discern
  GROUP BY pair HAVING cnt=2) AS t2
WHERE t1.pair=t2.pair GROUP BY t1.pair;
```

5.4 Quality Measures

One of popular methods of evaluating usefulness of attributes is to calculate how many pairs an attribute can discern. When some attributes are already chosen, then we should count only pairs which are not discerned yet.

Let X be a set of attributes already selected. Generating a measure value for the rest of attributes can be done as follows: get all pairs which are discerned by X; find all remaining pairs (using WHERE NOT EXISTS or an outer join with IS NULL); find all attributes discerning these pairs, weighted by a number of pairs an attribute can discern. An outer join version of SQL query generating all candidate attributes is presented below (by $..X..$ we denote an explicit list or a subquery generating attribute numbers from set X):

```
SELECT t1.a, COUNT(DISTINCT pair) AS weight
FROM discern AS t1 LEFT JOIN
(SELECT DISTINCT pair FROM discern WHERE a IN (..X..)) AS t2
ON t1.pair = t2.pair WHERE t2.pair IS NULL
GROUP BY t1.a ORDER BY weight DESC;
```

In [10] another attribute measure was proposed, calculated directly from the decision table stored as EAV and suitable especially for continuous values. Assume that t is a data table (e, a, v), and a decision value for every object e is stored in a separate table *decisions*:

```
SELECT a, STDDEV(avg_val) AS quality FROM
 ( SELECT t.a, t1.dec, avg(t.v) AS avg_val FROM t, decisions t1
   WHERE t.e = t1.e GROUP BY a, dec ) AS t2
GROUP BY a ORDER BY quality DESC;
```

```
  Input: discernibility table discern(pair, a)
    initial set of attributes in table candidates(set_id, a)
    where set_id is 0 for the initial set
  Intermediate table: intermediate(set_id, a)
  Output: additional attributes in table candidates

CREATE PROCEDURE bottom_up()
BEGIN
  REPEAT
    DELETE FROM intermediate;
    // Find and add the best attribute
    INSERT INTO intermediate
      SELECT 0, a_to_add FROM
        (SELECT t1.a AS a_to_add, COUNT(DISTINCT t1.pair) AS c
        FROM discern AS t1 LEFT JOIN
          (SELECT DISTINCT pair FROM discern
          WHERE a IN (SELECT a FROM candidates)) AS t2
        ON t1.pair = t2.pair WHERE t2.pair IS NULL
        GROUP BY t1.a ORDER BY c DESC LIMIT 1
      ) t3;
    INSERT INTO candidates SELECT * FROM intermediate;
  UNTIL (SELECT COUNT(*) FROM intermediate) = 0 END REPEAT;
END
```

Fig. 2. SQL procedure for generating a good superreduct by a greedy algorithm

6 SQL-Based Attribute Reduction

6.1 Bottom-Up Building of a Candidate Set of Attributes

A known method of attribute selection in data mining [3] is to start with a set of attributes (which may consists of core attributes, other arbitrary set or even be empty) and then to add the best (with respect to some measure) attributes one by one, until the resulting set is sufficient to discern all pairs of objects in the discernibility table.

We adopted an attribute weight given by the query presented at the end of Section 5 as a measure for greedy bottom-up construction of a possibly small set of attributes. In each step we are adding an attribute which discerns the largest amount of pairs which are not discerned yet. We stop when all pairs are discerned, as shown in Figure 2.

The above procedure results with a superreduct, often containing redundant elements. Thus, the attribute set stored in *candidates* must be reduced.

6.2 Top-Down Reduction of a Candidate Set of Attributes

The procedure presented in Figure 4 is a complete, working algorithm generating all reducts – subsets of a given candidate set. See also Figure 3 for an outline of a single step. As the resulting set of reducts (as well as computation time) can be exponential, it is advisable to start with good candidate sets ("nearly reducts"), e.g. found by a bottom-up procedure described in Section 6.1. An outline of this algorithm is as follows:

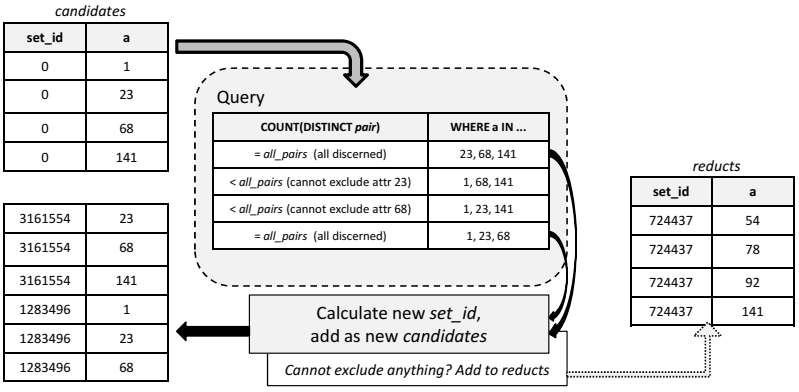

Fig. 3. A single step of the attribute reduction process

1. Select the first set of attributes from a queue (table *candidates*).
2. Use it to generate all superreducts shorter by a single attribute.
3. Insert these superreducts into *intermediate* table.
4. If the result of step 2 was empty, then the set chosen in step 1 is a reduct – insert it into an output *reduct* table (omit repetitions).
5. Add the new subsets from *intermediate* table into *candidates*.
6. Continue from step 1, until the *candidate* table is empty.

Steps 2 and 3 can be done by a single SQL query. Note that analyzing a whole family of subsets in a single query may provide performance advantages connected with internal caching mechanisms of a query engine.

The only nontrivial technique that was not described in earlier sections is a mechanism of ensuring uniqueness of subsets stored in *candidates* and *reducts*. In general, it is hard in SQL to operate on a set of rows and to compare it with another set. However, every subset of attributes in *candidates* and *reducts* can be labeled with an arbitrary integer identifier *set_id*. The initial candidate set may be labeled by 0, and any other set is numbered as $parent\_id\ XOR\ CRC32(a)$, where a is an attribute just excluded from the parent set. As binary XOR is symmetrical, each subset has a unique identifier, no matter what was the order of attribute exclusion which led to it (assuming $CRC32$ is strong enough as the hashing function). Figure 4 displays the complete algorithm.

7 Induction of Decision Trees

A set of good attributes (e.g. a reduct) may be used to generate a predictive data model in a form of decision rules or trees. An idea of using SQL-based solution in the process of a decision tree construction was presented in [14]. More specifically, the authors investigated the problem of minimizing the number of database SQL queries needed in order to find the binary partition of continuous attribute domain that is close to the optimal (with respect to a given quality measure) one.

```
  Input: discernibility table discern(pair, a)
    initial set of attributes in table candidates(set_id, a)
    where set_id is 0 for the initial set
    @all_pairs is equal to a number of discerned pairs of objects
  Intermediate table: intermediate(set_id, a)
  Output: table reducts(set_id, a), initially empty

CREATE PROCEDURE reduce()
BEGIN
  WHILE((SELECT COUNT(*) FROM candidates) ≠ 0) DO
    SET @cur_id = (SELECT MIN(set_id) FROM candidates);
    // Exclude single attributes, if possible:
    INSERT INTO intermediate
      SELECT (@cur_id XOR CRC32(a_excl)), a FROM candidates JOIN
      (SELECT candidates.a AS a_excl, COUNT(DISTINCT pair) AS cnt
        FROM discern, candidates WHERE
        discern.a IN (SELECT a FROM candidates WHERE set_id = @cur_id)
        AND discern.a ≠ candidates.a AND set_id = @cur_id
        GROUP BY candidates.a HAVING cnt = @all_pairs
      ) t1
      WHERE set_id = @cur_id AND a ≠ a_excl;
    // Reduct, if nothing was excluded:
    INSERT INTO reducts SELECT * FROM candidates
      WHERE set_id = @cur_id
      AND (SELECT COUNT(*) FROM intermediate) = 0
      AND set_id NOT IN (SELECT DISTINCT set_id FROM reducts);
    // Cleanup:
    DELETE FROM candidates WHERE set_id = @cur_id;
    INSERT INTO candidates SELECT * FROM intermediate
      WHERE set_id NOT IN (SELECT DISTINCT set_id FROM candidates);
    DELETE FROM intermediate;
  END WHILE;
END
```

Fig. 4. SQL procedure for generating all reducts which are subsets of a given set

The approach presented is highly dependent on the number of attributes due to the fact that the computations are performed for each attribute independently. A different approach presenting a procedure of decision tree induction, based on SQL and EAV form, was proposed in [10]. It was based on a greedy rule of choosing the most promising attribute from data. Then we create a cut on its range in order to possibly maximally separate objects from different decision classes. One of the main assets of using EAV model is used here: it enables to significantly limit number of queries by assigning the cuts for all nodes at a given level with a single SQL query. The algorithm consists of:

i Finding the best cut for the whole data set in the root;
ii For a tree of depth $i-1$, finding the best cuts producing nodes at depth i;
iii Stopping if a predefined depth k of the tree is reached.

We consider binary decision class to avoid introducing potentially complicated rule of cut creation for the chosen attribute. For attribute which was chosen using the measure from Section 5.4 we calculate the cut as the arithmetic average of all averages from decision classes.

The whole process consists of four database tables: the main table t in EAV format, the additional table *decisions* storing decision labels for objects, a

temporary table *tmp* for statistics in subgroups, and *distr* table for storing distribution (assignment) of objects to the deepest level of the tree. For a detailed description and complete SQL process please refer to [10].

8 Experimental Framework for Reduct Finding

8.1 A Data Set

The AAIA'2014 data set used in our experiments consists of 50,000 rows corresponding to reports from actions carried out by the Polish State Fire Service.[1] It has 11,852 sparse attributes and an imbalanced decision attribute. The original table (CSV source) has 1.1 GB while its discernibility counterpart occupies 40 GB, which is still feasible. It makes it a suitable case study for the SQL-based algorithmic framework proposed in this paper.

Let us also add that the data mining competition based on the AAIA'2014 data set has been aimed at searching for ensembles of possibly small attribute subsets that are able to cooperate with each other by means of locally induced classifiers. This is quite a difference comparing to other high-dimensional data mining competitions related primarily to classification accuracy, with no explicit evaluation of the stage of attribute selection [6,27].

8.2 Discernibility Sampling

Pairs of objects (which correspond to a single cell in discernibility table) are more important if they are hard to be discerned, i.e. if there is few attributes discerning them. Thus, to obtain a sample of discernibility table, it is reasonable to select the most important pairs.

The example below generates a 1% sample (100,000 pairs) from an original discernibility table (nearly 10,000,000 pairs) of AAIA'2014 data:

```
INSERT INTO disc_sample
SELECT t1.pair, t1.a FROM discern AS t1,
  (SELECT pair, COUNT(*) cnt FROM discern
   GROUP BY pair ORDER BY cnt LIMIT 100000) AS t2
WHERE t1.pair=t2.pair ORDER BY 2,1;
```

Such table does not reflect to any sampling of original rows, but it gathers all the pairs of rows which are hard to be discerned. If a set $R \subseteq A$ is a reduct of such *disc_sample*, then it is usually a subreduct of the original *discern*, but usually close enough to be a good base for bottom-up building of a candidate. On the other hand, if such R happens to be a superreduct of *discern*, then it is for sure a proper reduct. Checking discernibility (see Section 5) is much cheaper than ensuring that R is a reduct.

[1] `https://fedcsis.org/2014/dm_competition`

8.3 Performance Results

The AAIA'2014 data set was loaded into the EAV table and then transformed into the discernibility table as described in Section 4. The procedure of calculating reducts consisted of generating good candidates on a sample and then extending them using the whole discernibility table. For a sampled case, the bottom-up procedure takes 18 seconds, the top-down reduction takes about 40 seconds, and it produces a few reducts of length 15.

The bottom-up greedy procedure on the whole table takes 45-60 minutes and produces sets of about 20 attributes. The top-down reduction started from a superreduct found by the greedy algorithm takes about 30 minutes and produces one or more reducts usually consisting of 18 attributes. For comparison, the procedure started from a subset chosen randomly from 30 attributes with the best scores of previously discussed quality measure lasts 2-3 times longer and usually produces bigger reducts (19-20 attributes).

9 Conclusions

The presented ideas may be treated as a step toward understanding to what extent SQL-level interfaces can be useful when designing, implementing and performing KDD operations over large data sets, particularly in relation to rough set-based methods of attribute reduction.

We noted that it is possible to construct fast-performing scripts based on automatically generated analytic queries, given a proper choice of an underlying RDBMS technology. The obtained performance results suggest that it may be useful for database systems to introduce approximate query extensions [8]. Moreover, using SQL-based algorithms as a starting point, one can develop their more optimal realizations using lower-level languages [30].

In some cases, specialized implementations or solutions that use large computational clusters may outperform the presented SQL-based algorithms both in terms of the achieved performance or the time complexity of a model induction. However, this will usually require much more effort to implement them than to use the embedded features of the data storage system. Therefore, these two approaches cannot be considered as competing with each other but rather as complementary solutions. The same complementarity between SQL-based and non-SQL-based methodologies can be seen also for other data mining tasks [2] and complex deployments of data processing environments [20].

In future, we will continue working on aggregate SQL statements evaluating, e.g., multiple candidate subsets of attributes. It may be also interesting to consider more dynamic scenarios, where new attributes are extracted and added while running the attribute selection procedures [26,28]. Another important future aspect refers to the already-mentioned usage of approximate extensions of standard SQL language in the KDD tasks. Last but not least, we are intending to develop SQL-based procedures for learning more complex decision models from data such as, e.g., ensembles of reducts or decision forests [7].

References

1. Apanowicz, C., Eastwood, V., Ślęzak, D., Synak, P., Wojna, A., Wojnarski, M., Wróblewski, J.: Method and system for data compression in a relational database. US Patent 8,700,579 (2014)
2. Bae, S.-H., Choi, J.Y., Qiu, J., Fox, G.C.: High Performance Dimension Reduction and Visualization for Large High-dimensional Data Analysis. In: Proc. of HPDC, pp. 203–214 (2010)
3. Bazan, J.G., Nguyen, H.S., Nguyen, S.H., Synak, P., Wróblewski, J.: Rough Set Algorithms in Classification Problem. In: Rough Set Methods and Applications, pp. 49–88. Physica-Verlag (2000)
4. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In: OSDI, pp. 137–150 (2004)
5. Hu, X., Han, J., Lin, T.Y.: A New Rough Sets Model Based on Database Systems. Fundamenta Informaticae 59(2-3), 135–152 (2003)
6. Janusz, A., Nguyen, H.S., Ślęzak, D., Stawicki, S., Krasuski, A.: JRS'2012 Data Mining Competition: Topical Classification of Biomedical Research Papers. In: Yao, J., Yang, Y., Słowiński, R., Greco, S., Li, H., Mitra, S., Polkowski, L. (eds.) RSCTC 2012. LNCS, vol. 7413, pp. 422–431. Springer, Heidelberg (2012)
7. Janusz, A., Ślęzak, D.: Rough Set Methods for Attribute Clustering and Selection. Applied Artificial Intelligence 28(3), 220–242 (2014)
8. Kowalski, M., Ślęzak, D., Synak, P.: Approximate Assistance for Correlated Subqueries. In: Proc. of FedCSIS, pp. 1455–1462 (2013)
9. Kowalski, M., Ślęzak, D., Toppin, G., Wojna, A.: Injecting Domain Knowledge into RDBMS – Compression of Alphanumeric Data Attributes. In: Kryszkiewicz, M., Rybinski, H., Skowron, A., Raś, Z.W. (eds.) ISMIS 2011. LNCS, vol. 6804, pp. 386–395. Springer, Heidelberg (2011)
10. Kowalski, M., Stawicki, S.: SQL-Based Heuristics for Selected KDD Tasks over Large Data Sets. In: Proc. of FedCSIS, pp. 303–310 (2012)
11. Kuncheva, L.I.: Combining Pattern Classifiers: Methods and Algorithms. Wiley (2004)
12. Liu, H., Motoda, H. (eds.): Feature extraction, construction and selection – a data mining perspective. Kluwer Academic Publishers, Dordrecht (1998)
13. Liu, H., Motoda, H.: Computational Methods of Feature Selection. Chapman & Hall/CRC (2008)
14. Nguyen, H.S., Nguyen, S.H.: Fast split selection method and its application in decision tree construction from large databases. Int. J. Hybrid Intell. Syst. 2(2), 149–160 (2005)
15. Nguyen, H.S., Ślęzak, D.: Approximate reducts and association rules. In: Zhong, N., Skowron, A., Ohsuga, S. (eds.) RSFDGrC 1999. LNCS (LNAI), vol. 1711, pp. 137–145. Springer, Heidelberg (1999)
16. Pawlak, Z., Skowron, A.: Rudiments of Rough Sets. Information Sciences 177(1), 3–27 (2007)
17. Rahman, M.M., Ślęzak, D., Wróblewski, J.: Parallel Island Model for Attribute Reduction. In: Pal, S.K., Bandyopadhyay, S., Biswas, S. (eds.) PReMI 2005. LNCS, vol. 3776, pp. 714–719. Springer, Heidelberg (2005)
18. Sarawagi, S., Thomas, S., Agrawal, R.: Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications. Data Min. Knowl. Discov. 4(2/3), 89–125 (2000)

19. Ślęzak, D., Kowalski, M.: Towards approximate SQL – infobright's approach. In: Szczuka, M., Kryszkiewicz, M., Ramanna, S., Jensen, R., Hu, Q. (eds.) RSCTC 2010. LNCS, vol. 6086, pp. 630–639. Springer, Heidelberg (2010)
20. Ślęzak, D., Stencel, K., Nguyen, H.S.: (No)SQL Platform for Scalable Semantic Processing of Fast Growing Document Repositories. ERCIM News 2012(90) (2012)
21. Ślęzak, D., Synak, P., Wojna, A., Wróblewski, J.: Two Database Related Interpretations of Rough Approximations: Data Organization and Query Execution. Fundamenta Informaticae 127(1-4), 445–459 (2013)
22. Ślęzak, D., Wróblewski, J., Eastwood, V., Synak, P.: Brighthouse: An Analytic Data Warehouse for Ad-hoc Queries. PVLDB 1(2), 1337–1345 (2008)
23. Świeboda, W., Nguyen, H.S.: Rough Set Methods for Large and Spare Data in EAV Format. In: Proc. of RIVF, pp. 1–6 (2012)
24. Szczuka, M.S., Wojdyłło, P.: Neuro-wavelet classifiers for EEG signals based on rough set methods. Neurocomputing 36(1-4), 103–122 (2001)
25. Widz, S., Ślęzak, D.: Rough Set Based Decision Support – Models Easy to Interpret. In: Selected Methods and Applications of Rough Sets in Management and Engineering, pp. 95–112. Springer (2012)
26. Widz, S., Ślęzak, D.: Granular attribute selection: A case study of rough set approach to MRI segmentation. In: Maji, P., Ghosh, A., Murty, M.N., Ghosh, K., Pal, S.K. (eds.) PReMI 2013. LNCS, vol. 8251, pp. 47–52. Springer, Heidelberg (2013)
27. Wojnarski, M., et al.: RSCTC'2010 Discovery Challenge: Mining DNA Microarray Data for Medical Diagnosis and Treatment. In: Szczuka, M., Kryszkiewicz, M., Ramanna, S., Jensen, R., Hu, Q. (eds.) RSCTC 2010. LNCS, vol. 6086, pp. 4–19. Springer, Heidelberg (2010)
28. Wróblewski, J.: Analyzing relational databases using rough set based methods. In: Proc. of IPMU, vol. 1, pp. 256–262 (2000)
29. Wróblewski, J.: Pairwise Cores in Information Systems. In: Proc. of RSFDGrC, vol. 1, pp. 166–175 (2005)
30. Zhang, J., Li, T., Ruan, D., Gao, Z., Zhao, C.: A parallel method for computing rough set approximations. Information Sciences 194, 209–223 (2012)