ISBN 978-83-60810-51-4

SQL-Based Heuristics for Selected KDD Tasks over Large Data Sets

Marcin Kowalski* and Sebastian Stawicki*
*Institute of Mathematics, University of Warsaw
ul. Banacha 2, 02-097 Warsaw, Poland
{mkowal, stawicki}@mimuw.edu.pl

***Abstract*—We investigate how to use the scripts with automatically generated fast-performing analytic SQL statements to speed up the KDD-related tasks of attribute selection and decision tree induction. We base our framework on the entity-attribute-value data model in order to seamlessly scale the required queries with respect to the amounts of attributes involved in the given task's specification. We note that the considered tasks can be heuristically handled using the same class of aggregation queries, where the most promising attributes and splits are searched by analyzing diversity of aggregated results grouped by decision. We also outline our plans with respect to creation of a large-scale framework for evaluating the proposed heuristics against real-world data.**

***Keywords*-attribute selection; decision trees; SQL; EAV;**

I. Introduction

The PROCESS of knowledge discovery in databases (KDD) consists of many stages, such as data integration and cleaning, feature extraction, dimension reduction, model construction and more [1], [2]. Given the growing sizes of data sets, there are a number of approaches attempting to mine the available sources by means of database query languages rather than operating directly on data [3], [4], [5], [6]. Due to its simplicity and commonness, using SQL for such purposes is probably the most examined and described in the literature [7], [8], [9].

One of the major trends in described area is to split the main KDD task into subtasks solved by relatively basic SQL queries (e.g. [8]), the other trend is to precalculate tables containing some partial results frequently used in implemented algorithms' runs. The presented work is embedded in both of these trends although the considered examples of KDD algorithms follow a more iterative, ad-hoc style of formulating and executing SQL statements. When ordinary SQL statements that are available in most RDBMS-s became not enough - especially in the case of growing compoundness of tasks - researches reach beyond the contemporary language standard and utilize extensions of SQL such as user defined functions (including aggregators) or non-scalar types support [8]. In this paper we concentrate on methods based on standard SQL's operands and aggregations provided by most RDBMS vendors. We do not actually try to implement known algorithms from KDD domain but rather adapt them to the proposed model and replace compound calculations by simple procedures that can be easily calculated with plain SQL queries.

We choose three examples of classical problems from the domain and formulate them as SQL supported tasks [10], [11]. We focus our attention on the decision tree induction but we also describe potential applications of the presented approach in the attribute selection and extraction problems. We show that a significant part of SQL statements used in all these cases has the same general structure which may be helpful in order to choose an appropriate RDBMS solution.

We redefine the selected tasks stages using analytic SQL over data sets stored in an entity-attribute-value (EAV) format. We claim that even the previously known SQL-based techniques of attribute selection and decision tree induction can be improved by means of establishing a more scalable framework. In particular, we show that the EAV layout enables to design SQL-based scripts independently from the amounts of attributes, which is not the case for implementations utilizing a standard tabular format.

For the sake of simplicity, all methods described in this paper are designed for decision tables with numerical attributes. However they might be adapted for all types of data.

The paper is organized as follows: Section II recalls the EAV model of data and introduces a simple measure on attributes that we utilize for mining EAV data sets. In Section III, our SQL-based heuristics for the decision tree induction problem is outlined. Section IV presents experimental validation of method introduced in Section III compared to one of the state-of-the-art algorithms for learning decision tree. Section V extends the proposed methodology onto the other classical problems from the area of KDD, with a special emphasis on feature selection and extraction. Section VI concludes the paper and points out directions to be studied in the future.

II. Entity-Attribute-Value Approach

The entity-attribute-value (EAV) stores and their extensions have recently got attention as a universal means to process large amounts of heterogeneous data [12], [13]. The advantages of EAV have been already studied in knowledge discovery and data mining [6], [14]. If combined with powerful enough database technologies, it raises an opportunity to rewrite in a more flexible form some of the already existing SQL-based data mining approaches that were originally designed for data tables where columns corresponded directly to the attributes specified in a learning task. Surely, it is even

TABLE I
A DECISION TABLE IN A STANDARD TABULAR FORM.

U	a_0	a_1	a_2	a_3	d
1	1	101	50	36	0
2	8	176	90	300	1
3	7	150	66	342	0
4	7	187	68	304	1
5	0	100	88	110	0
6	0	105	64	142	0
7	1	95	66	38	0

TABLE II
A DECISION TABLE IN AN EAV FORMAT.

object	*attribute*	*value*
1	a_0	1
1	a_1	101
1	a_2	50
1	a_3	36
1	d	0
2	a_0	8
2	a_1	176
2	a_2	90
⋮	⋮	⋮
6	d_0	0
6	d_1	105
6	d_2	64
6	d_3	142
6	d	0
7	d_0	1
7	d_1	95
7	d_2	66
7	d_3	38
7	d	0

TABLE III
A DECISION TABLE IN AN EAV FORMAT WITH ISOLATED DECISION.

object	*attribute*	*value*	*d*
1	a_0	1	0
1	a_1	101	0
1	a_2	50	0
1	a_3	36	0
2	a_0	8	1
2	a_1	176	1
2	a_2	90	1
⋮	⋮	⋮	
6	a_0	0	0
6	a_1	105	0
6	a_2	64	0
6	a_3	142	0
7	a_0	1	0
7	a_1	95	0
7	a_2	66	0
7	a_3	38	0

more interesting for the KDD-related tasks, for which SQL-based implementations were not studied yet.

Table I displays a decision table $\mathbb{A} = (U, A \cup \{d\})$, where U contains eight objects, A contains three attributes, and decision d takes a form of column Flu. Table II displays the same data in an EAV form. Each object (or a row) is stored as a series of triples - each triple consists of three things: identifier of the object, identifier of an attribute and the actual value of the attribute for the object. The reminder of this section outlines a simple data mining algorithm rewritten in a form supported by queries executed against EAV. In the next sections, analogous techniques are used for other tasks.

Assume that our EAV table was created with the following schema:

```
create table EAV (
    obj BIGINT,
    attr INT,
    val INT);
```

We introduce a measure which evaluates attributes and expresses their usefulness according to a simple heuristic approach. Namely, let us consider a measure computed using a single SQL statement that evaluates usefulness of all attributes based on analysis of diveristy of their values' averages in particular decision classes.

Indeed, there is an intuition that an attribute with maximally diversed means for particular decision classes provides the highest degree of information about the decision. Surely, this is just a purely heuristic approach but its major advantage is the speed of execution, especially for the RDBMS solutions designed for such types of aggregations. From theoretical perspective, such a measure seems to follow the principles of Bayesian data analysis where attributes and their values are evaluated subject to possible decision classes [15], [16].

One simple realization of such kind of a function may be a standard deviation of means of attribute's values within decision classes. If the standard deviation is high we may

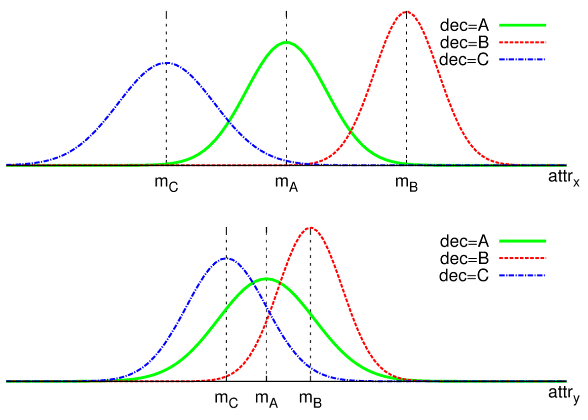

Fig. 1. Intuition of introduced measure - distributions (and means) of attribute values in particular decision classes. Examples of: easier separable classes for "useful" attribute (top - $attr_x$) and harder separable classes - less "useful" attribute (bottom - $attr_y$)

suspect that the attribute can (with a high degree) distinguish objects with different decisions. Figure 1 displays intuition: top chart presents situation with attribute of good 'quality' where decision classes are relatively easy separable; bottom chart presents a kind of opposite situation.

Here are simple SQL queries that arranges attributes w.r.t. their usefulness understood as above. The first query is as follows

```
select attr, stddev(avg_val) as quality from
   (
   select v.attr, l.dec, avg(v.val) 'avg_val'
   from EAV v, LABELS l
   where v.obj = l.obj
   group by attr, dec
   ) attr_avg_values
group by attr
order by quality desc;
```

for database schema related to format similar to presented in Table III with separate `LABELS` table containing assignment of each object to decision class (the table contsists of 2 columns: `(obj INT, dec INT)`). The second SQL statement to be considered is the following

```
select attr, stddev(avg_val) from
   (
   select v.attr, d.val, avg(v.val) 'avg_val'
   from EAV v, EAV d
   where v.obj=d.obj and d.attr=dec
   group by 1,2
   ) attr_avg_values
group by attr;
```

for table schema resembling the one presented in Table II containing decision attribute in `EAV` table (`dec` is a literal which identifies the decision attribute). The presented queries are just exemplary ones and can be adapted to customized model schema.

III. Decision Tree Induction

Decision Tree Induction implemented with SQL support is one of the most studied tasks in the presented area. The main researchers' efforts are aimed here at optimization of well known and verified algorithms known from the machine learning literature [17], [18]. Usually it is done by applying techniques to minimize number of submitted queries [19]. Other methods base on limitation of number of redundant calculations by precalculation of some partial results intesively used during the main algorithm's run [8]. Our approach, as mentioned in Section II is based on simple rule of choosing most promising attribute from data. Then using the method 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.

Our approach of inducing decision tree based on SQL is iterative and consists of:

i Finding the best cut for the whole data set in the root.
ii For a given tree of depth $i-1$ - finding the best cuts for all nodes on 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 chosen attribute. For the sake of simplicity, for attribute which was chosen using the measure from Section II we calculate the cut as arithmetic average of all averages from decision classes.

In this paper we skip a pruning step of decision tree creation process, exposing the method to produce an overlearned classifier. The pruning can also be implemented in SQL manner for instance by adopting some SQL-based methods of extracting association and decision rules [20], [6]. We are going to face this problem in the near future.

We use three following database tables:

- main table, consists whole decision table (information table) in EAV format; `dec` stands for decision attribute and - as a column - is redundant (table `EAV` may be normalized in real life - like shown in one of the cases in previous section). It is done purposely - to make the method description short. In general situation `val` column can be string column and modification of the rule presented in would be necessary, but for the sake of simplicity, we present integer column here. Schema of `EAV` table is shown below:
```
create table EAV (
    obj BIGINT,
    attr INT,
    val INT,
    dec INT );
```
- table `CUTS` consists information about all tests and nodes in decision tree; each row represents one node of the tree with respective cut; after each iteration new rows (one for each new node) are added into it. In presented approach one searches for tests of a "greater than" form $attr > c$, where c stands for constant. For example a row `(3, 23, -2.78)` in the table means that for a node of id equals to 3 we should check if objects fulfill $a_{23} > -2.78$ predicate. Values of the $node\_id$ column fulfills the following conditions:
  - the id of the tree root node is equal to 1
  - descendants of a node identified by i have their ids equal to $2i$ (for objects that do not pass the test from the parent node) and $2i+1$ (otherwise) respectively.

  Table `CUTS` is built with the following schema:
```
create table CUTS (
    node_id INT,
    attr INT,
    cut DOUBLE );
```
- Distribution table containing assignments of each object (row in original decision table) to the tree node on the lowest level (deepest) of the tree. This table is actualized and rewritten after each iteration.
```
create table DISTR (
    obj BIGINT,
    node_id INT );
```

At the beginning of the tree building process all objects are assigned to the root node.

Induction step. In each iteration there are best cuts calculated using described criterion in subgroups labelled by ids of nodes of the deepest level of tree built so far. Using one SQL query we can calculate next best cuts for all nodes in tree at a given depth.

The measure described in Section II is suitable for any number of decision classes. As pointed out earlier we used here a simplified version of the proposed heuristic. Since our

task here is the binary classification, instead of the standard deviation of mean values of attributes with respect to the decision classes, we can choose the most promising attribute examining just the distance between the mean values. One can easily show the equivalence of the two approaches.
Each iteration of the algorithm consists of 3 steps:

1) create temporary table with chosen statistics in subgroups defined by nodes of the tree built so far. This table is also rewritten after each iteration:

```
create table TMP as
select d.node_id, e.attr,
avg(case when e.dec=1 then e.val
  else NULL end) a1,
avg(case when e.dec=0 then e.val
  else NULL end) a0
from EAV e, DISTR d
where e.obj = d.obj
group by 1, 2;
```

2) calculate best cuts w.r.t. measure from Section II for all leaves of tree built so far. As mentioned earlier, we reformulate criterion of maximal standard deviation of averages in decision classes and decided for the arithmetical average of calculated averages as the cut selection rule. Results are automatically insert into tables `CUTS`:

```
insert into cuts
select  s.node_id, s.attr, (a1+a0)/2 cut
from (
  select t.node_id, max(abs(t.a1-t.a0)) m
  from TMP t
  group by 1) x
join TMP s on
x.node_id = s.node_id and
x.m = abs(s.a1-s.a0)
where  a0 IS NOT NULL and a1 IS NOT NULL;
```

3) assign each object into just calculated new nodes and rewrite table `DISTR`

```
create table distr as
select d.obj,
       2*d.node_id + (e.val > c.cut)
       as node_id
from DISTR d, EAV e, CUTS c
where e.obj = d.obj and
c.attr = e.attr and
c.node_id = d.node_id
order by d.obj
```

Analysis. To find cuts for all nodes at any level of tree depth we need 3 queries. Despite all of them requiring I/O operation on disk, we find them not too demanding for reasonably sized tree. Number of such operations depends on: product of actual tree depth and number of attributes (for `TMP` table), tree depth (for `CUTS`), number of objects (for `DISTR`). Presented queries are quite straightforward and do not exceed syntax of standard GROUP BY aggregations, so can be executed in most of relational database engines.

The main bottleneck of the presented approach may be requirement of storing large number of rows in one table (`EAV` table case). According to our experiments, this approach may be applied to data sets sized of billions rows in `EAV` table (e.g. of an order of 10^6 objects and 10^4 attributes). However it may depend on database engine. It is quite the same in EAV model if there exists advantage of number of objects over number of attributes or vice versa. The approach seems to be flexible in this matter.

IV. Experiments

In order to test the soundness of the proposed approach we performed experiments related to building decision trees comparing the method and some state-of-the-art algorithms implemented in R-system [21]. We chose data from one of the KDD contest [22] related to classification documents into topics containing 10000 objects and 25640 conditional attributes (all were numeric and rare). While the contest referred to a multi-label classification problem, we decided to choose 2 labels with the highest prior probability in the training set. We considered separately 2 binary classification problems for 2 separate decision attributes (decision attributes with labels: 40 and 44 from the data set). We have split the training data set (for which true labels were available) into two subsets of 5000 objects each and on first of them we applied tree building algorithm. Both data sets were transformed into EAV format. Our `EAV` table consisted of 128.2 million of rows.

We have chosen an open source database engine provided by Infobright [23] which stores data in compressed form. The input `EAV` table consumed only 2.1MB space on disk.

The second subset was used to validate built tree with simple rule of assigning decision class if probability in the node exceeded its prior probability [15], [16].

Tests were performed on machine with Intel Core i7 870 2.93GHz processor and 8GB RAM. Unfortunately, using community edition of chosen database engine, we were limited to only 1 core for calculations.

TABLE IV
AN EXEMPLARY TREE IN TABULAR FORMAT (`CUTS` TABLE) CALCULATED FOR DECISION $d40$ WITH MAXIMAL DEPTH = 4

node_id	attr	cut
1	2083	67.26111644000000
2	21657	48.38156421633018
3	21657	55.33525309917354
4	10371	44.46231628655018
5	7988	172.96122647631450
6	21895	49.59733955365024
7	22757	239.05555555555550

In Table IV there is presented an exemplary tree built by the SQL-based algorithm for decision $d40$ with limit of depth set to 4 (3 levels of tests). Tree in tabular form consists of triples containing `node_id`, id of the attribute which would be tested in the node (`attr`) and the cut value (`cut`). Figure 2 presents the exemplary tree in a graphical form.

On mentioned machine calculating single level of nodes during algorithm's run (one iteration of it) lasted on average less than 1.5 minutes. Details are in Figure 4.

In Figure 3 and Table V we present results from classification of test data set for various depth of built tree (various algorithm's stop criterion).

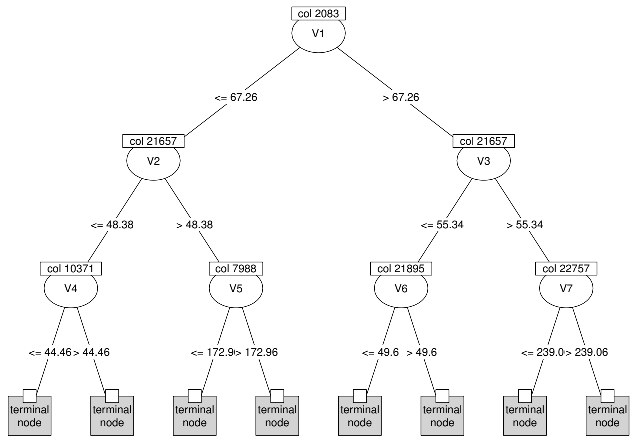

Fig. 2. Graphical representation of an exemplary tree calculated for decision $d40$ with maximal depth = 4

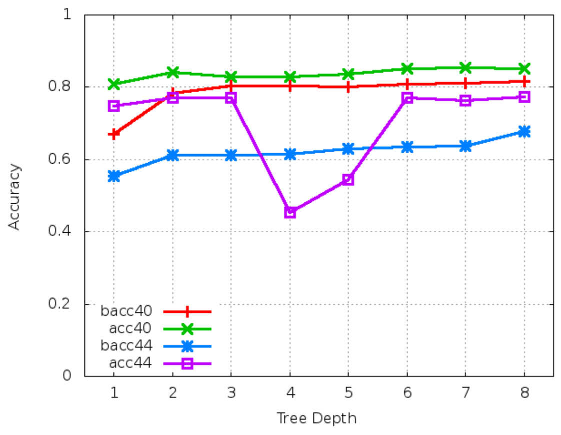

Fig. 3. Results of experiments for 2 decision attributes ($d40$ and $d44$): relationship between accuracy (acc), balanced accuracy (bacc) of classifiers for test data set and maximal depth tree. See Table V for details.

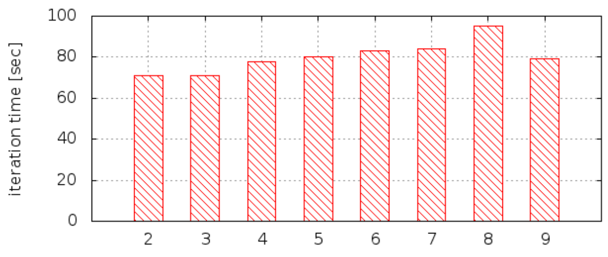

Fig. 4. Duration in seconds of consecutive iterations

TABLE V
RESULTS OF DECISION TREE INDUCTION USING SQL.

tree depth	bacc40	acc40	bacc44	acc44
8	0.8154	0.8518	0.6769	0.7726
7	0.8112	0.8530	0.6380	0.7642
6	0.8093	0.8522	0.6350	0.7704
5	0.8021	0.8366	0.6301	0.5450
4	0.8035	0.8292	0.6138	0.4526
3	0.8034	0.8290	0.6118	0.7704
2	0.7822	0.8424	0.6118	0.7704
1	0.6709	0.8098	0.5553	0.7488

From the Figure 3 and Table V we can observe some unstable behaviour for accuracy measure and decision attribute $d44$. It is caused by existence of heavy leaf in the tree with probability of decision attribute equals to 1 being close to the prior probability. If the maximal depth of the tree is equal to 4 and 5 this leaf is classified to "wrong" decision class. Situation normalized after depth 6 is reached. Then some observations are cut off the node and mentioned probability falls below the prior probability and in consequence assigned decision class is changed.

We have performed experiments in R-system to assess reference results. We have used the `rpart` library to create decision tree model. Unfortunately for the standard model settings because of insufficient system resources model creation process was unsuccessful. Recursive partitioning and regression trees package (`rpart`) turn out to be sensitive to large values of numbers of attributes. Package `party` showed similar behaviour. We decided to decrease number of attributes and build the decision trees limiting the data to the first 10000 attributes. This approach seemed as a good solution for the arisen situation as we wanted to do only a rough results comparison. Finally, model creation on 5000 training samples limited to the first 10000 attributes for decisions $d40$ and $d44$ took about 9 minutes each. The model performance were measured on the test 5000 samples and expressed by accuracy and balanced accuracy. The results were as follows:

TABLE VI
RESULTS OF DECISION TREE INDUCTION USING R-PROJECT.

R-system - `rpart` library	bacc	acc
10000_attrs_dec40	0.7906	0.8308
10000_attrs_dec44	0.7056	0.8080

Despite discussed unsteadiness we find presented results of using SQL in decision tree induction satisfactory. Comparing to partial results - full were unobtainable - from non-tuned algorithm from R-project the results of proposed method do not differ significantly (we can even say that they were competitive). For sure, more tuning to the benchmark algorithm would increase its quality but we believe that also results of our method may be improved. Ways of achieving this may be e.g. changing cut selection criterion from average-based to the median-based or adding pruning stage to the algorithm. We are aware of potential prolongation of algorithm execution time when such improvements done by SQL queries are introduced but we believe the gain of classification quality will compensate it. The other direction we are considering promising is utilize the size of the node in cut selection criterion.

V. Attribute Selection And Attribute Extraction

In this section we consider the problem of attribute selection, which is recognized as one of fundamental tasks of KDD [1], [11]. Let us start with the simplest possible case where the goal is to score attributes $a \in A$ with respect to a degree of information they provide for the values of d. There are plenty of more or less sophisticated measures developed in the literature for this purpose. On the other hand, for large-sized real-world data the simplest solutions are often the best ones. Hence, we can one more time make use of the measure proposed in Section II. But in contrast to Section III, in this we present only ideas of the adapted algorithm.

In [6], [20], it was shown how to redesign the well-known *Apriori* algorithm for searching for association and decision rules using script of iteratively generated SQL statements involving massive self-joins and aggregations. The ideas for searching for if-then rules satisfying the constraints specified for their support and accuracy may be adapted also for the task of attribute selection. The mentioned support and accuracy should be of course replaced with average support and accuracy for some classifiers, e.g., decision rules created from evaluated attribute subsets.

The recalled algorithm and its ideas may be actually used to design also the attribute selection framework that would be analogous to the decision rule generation. Namely, the measure from Section II can be used to fill in the table of single candidate attributes analogous to filling in the initial table with the attributes that provide high enough level of heuristically computed information about the decision. It may be also not so difficult to imagine an analogy in generation candidates in both cases. For instance, one may consider the standard deviation of the values of a particular attribute as a kind of estimate of its ability to produce well-supported decision rules when constructing a classifier in the next stage of the KDD process. Further steps of generating attribute subset candidates (like in Apriori algorithm) may also utilize some modifications of the heuristic (e.g. Monte Carlo measures) in assessing quality of the attribute subsets.
The outline of the algorithm (similar to the *Apriori* algorithm) is as follows:

i The initial step in which we select single columns as initial candidates for the feature selection task.
ii Iterative two-step process of obtaining the best attribute subsets. Each step focuses on attribute subsets of the fixed size. Starting from single columns candidates and combining them to larger attribute subsets (enlarging the size of candidates by 1 on each step).
   - Candidates generation. Attribute subsets that were tested and recognized as promising in the previous step are transformed into larger candidates.
   - Candidates evaluation. Using the evaluation procedure to assess the candidates. We have tree possibilities for the candidate:
      1) It is good enough and can be regarded as a part of the overall result.
      2) It is poor and can be omitted from further consideration.
      3) It is promising and will be transfered to the next step.

All the above steps can be expressed by means of SQL statements.

Before proceeding with further analogies between the recalled framework for searching for decision rules satisfying the constraints specified with their support and accuracy and the task of attribute selection, let us note that one may extend this study onto some new, potentially better attributes that are derived from the original ones. In [2], one may find a number of practical hints how to extract a useful decision table, by means of both objects and attributes, from available data sources stored in a fully relational or partially unstructured form. Extraction of attributes is especially important in the case of high-dimensional data sets, such as biomedical or text data [13], [24]. In such cases, it would be optimal to evaluate the candidate features with no need of their explicit materialization or, at least, with no need of materializing too many of them. In [25], a genetic algorithm searching for new attributes defined as linear combinations of original attributes has been suggested. There was also considered how to build more sophisticated attributes from a multi-table relational data model. Let us focus on the simplest case of linear combinations of numeric attributes. In this case, one needs an appropriate fitness function evaluating particular chromosomes that encode new attributes. The above-discussed way of evaluating attributes by basing on the results of aggregate queries may be applied to this framework as well. There is no difficulty in designing a single SQL statement that would compute the average-value-based statistics for all formulas describing new attributes occurring in a given population of the genetic algorithm. It is enough to create an additional table that encodes such formulas in its rows and to properly join it while computing the aggregates.

Now, let us note that, regardless of whether we operate only with original attributes or we extend the search space onto their combinations, the key point is to search for the subsets of attributes that complement each other while describing the decision over the available data set. This problem – sometimes called the attribute subset selection in order to distinguish it from a far simpler task of searching for single attributes – is very thoroughly studied within the theory of rough sets, by means of so called decision reducts and approximate decision reducts [26], [27]. An (approximate) decision reduct is an irreducible subset of attributes that provide (almost) the same level of information about decision as the whole initial set of attributes. The level of information may be described by a measure that estimates an ability to train an accurate classifier by basing on the attributes in a given subset. In [28], it was shown that the degrees of information provided by a subset of attributes can be computed as the expected confidence of decision rules based on those attributes. Also, it was shown that subsets of attributes can be evaluated by means of the expected support of such decision rules. This means that the whole framework mentioned above could be adapted to the search of attribute subsets that approximately determine the decision classes and that correspond to classifiers that are general enough in order to expect them working well over the previously unseen cases. The only remaining task is to express such expected confidence and support measures using a simple, fast-performing SQL.

It is worth to mention also about future possibilities of using different measures for assessing attribute subsets. An interesting method that is focused on numerical attributes is presented in [29]. It searches through the possible binary cuts on attributes contained in the evaluated attribute subset and estimates the quality of classifier of chosen types constructed on top of the discretized attributes.

VI. Conclusions and Future Work

We outlined some analogies in building SQL-based heuristics aimed at searching for new attribute subsets and decision trees in large data sets stored in a triple-store-like format. 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. We tested method of decision tree induction using proposed criterion on one of the KDD contest's data sets and achieve satisfactory classification results comparing to the state-of-the-art decision tree induction methods. Some potential directions of its implementation improvement were also discussed.

In the future, we are going to implement and perform experiments for the attribute selection ideas outlined in Section V. We intend to extend our SQL-based framework onto learning ensembles of decision models, such as decision forests or ensembles of attribute subsets [10], [27]. The problem of adapting solution to the categorical attributes needs to be investigated as well. We also intend to take an advantage of some modern extensions of standard RDBMS functionality, such as the approximate OLAP and SQL methods in order to further speed up our algorithms. Last but not least, we are going to take a closer look at the experimental verification of our framework, which may be understood at the two following levels: verification whether SQL-based heuristics lead toward the same or similar output models as in the case of standard methods (e.g.: whether attribute subsets obtained using the proposed SQL scripts are significantly different than those obtained using algorithms working directly against data), and – in the case of differences at the first level – whether the quality of the obtained models is significantly worse.

We plan also to compare the presented methods with rough set methods implemented in, e.g. RSES [30], Rseslib [31], WEKA [32].

Acknowledgments

This work was partially supported by the Polish National Science Centre grant 2011/01/B/ST6/03867.

References

[1] W. Klösgen and J. M. Żytkow, Eds., *Handbook of Data Mining and Knowledge Discovery*. Oxford University Press, 2002.

[2] H. Liu and H. Motoda, Eds., *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Kluwer Academic Publishers, 1998.

[3] M. Mehta, R. Agrawal, and J. Rissanen, "SLIQ: A fast scalable classifier for data mining," in *EDBT*, 1996, pp. 18–32.

[4] S. Chaudhuri, "Data mining and database systems: Where is the intersection?" *IEEE Data Eng. Bull.*, vol. 21, no. 1, pp. 4–8, 1998.

[5] C. Ordonez and P. Cereghini, "SQLEM: Fast clustering in SQL using the EM algorithm," in *SIGMOD Conference*, 2000, pp. 559–570.

[6] S. Sarawagi, S. Thomas, and R. Agrawal, "Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications," *Data Min. Knowl. Discov.*, vol. 4, no. 2/3, pp. 89–125, 2000.

[7] S. Chaudhuri, U. M. Fayyad, and J. Bernhardt, "Scalable classification over SQL databases," in *ICDE*, 1999, pp. 470–479.

[8] K.-U. Sattler and O. Dunemann, "SQL database primitives for decision tree classifiers," in *CIKM*, 2001, pp. 379–386.

[9] H. S. Nguyen, "On efficient construction of decision trees from large databases," in *Rough Sets and Current Trends in Computing*, 2000, pp. 354–361.

[10] T. G. Dietterich, "An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization," *Machine Learning*, vol. 40, no. 2, pp. 139–157, 2000.

[11] I. Guyon et al., *Feature Extraction: Foundations and Applications*, ser. Studies in Fuzziness and Soft Computing. Springer, August 2006.

[12] D. J. Abadi, A. Marcus, S. Madden, and K. Hollenbach, "SW-Store: A Vertically Partitioned DBMS for Semantic Web Data Management," *VLDB J.*, vol. 18, no. 2, pp. 385–406, 2009.

[13] D. Ślęzak, A. Janusz, W. Świeboda, H. S. Nguyen, J. G. Bazan, and A. Skowron, "Semantic Analytics of PubMed Content," in *USAB*, ser. LNCS, vol. 7058, 2011, pp. 63–74.

[14] W. Świeboda and H. S. Nguyen, "Rough Set Methods for Large and Spare Data in EAV Format," in *RIVF*. IEEE, 2012, pp. 1–6.

[15] G. Box and G. Tiao, *Bayesian Inference in Statistical Analysis*. Wiley, 1992.

[16] D. Ślęzak, "Rough Sets and Bayes Factor," vol. 3400, pp. 202–229, 2005.

[17] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[18] T. M. Mitchell, *Machine learning*, ser. McGraw Hill series in computer science. McGraw-Hill, 1997.

[19] H. S. Nguyen and S. H. Nguyen, "Fast Split Selection Method and Its Application in Decision Tree Construction from Large Databases," *Int. J. Hybrid Intell. Syst.*, vol. 2, no. 2, pp. 149–160, 2005.

[20] D. Ślęzak and H. Sakai, "Automatic Extraction of Decision Rules from Non-deterministic Data Systems: Theoretical Foundations and SQL-Based Implementation," in *DTA*, ser. CCIS, vol. 64. Springer, 2009, pp. 151–162.

[21] http://www.r-project.org/, The R Project for Statistical Computing.

[22] http://sist.swjtu.edu.cn/JRS2012/ArticleCmd.aspx?AID=97, JRS 2012 Data Mining Contest.

[23] http://www.infobright.org/, Infobright Community Edition - relational database engine.

[24] M. Wojnarski, A. Janusz, H. S. Nguyen, J. G. Bazan, C. Luo, Z. Chen, F. Hu, G. Wang, L. Guan, and H. Luo, "RSCTC'2010 Discovery Challenge: Mining DNA Microarray Data for Medical Diagnosis and Treatment," in *RSCTC*, ser. LNAI, vol. 6086. Springer, 2010, pp. 4–19.

[25] J. Wróblewski, "Analyzing Relational Databases Using Rough Set Based Methods," in *IPMU, part 1*, 2000, pp. 256–262.

[26] M. J. Moshkov, M. Piliszczuk, and B. Zielosko, "On Construction of Partial Reducts and Irreducible Partial Decision Rules," *Fundam. Inform.*, vol. 75, no. 1-4, pp. 357–374, 2007.

[27] J. Wróblewski, "Ensembles of Classifiers Based on Approximate Reducts," *Fundam. Inform.*, vol. 47, no. 3-4, pp. 351–360, 2001.

[28] D. Ślęzak, "Rough Sets and Functional Dependencies in Data: Foundations of Association Reducts," *LNCS Transactions on Computational Science V*, pp. 182–205, 2009.

[29] D. Ślęzak and P. Betliński, “A Role of (Not) Crisp Discernibility in Rough Set Approach to Numeric Feature Selection,” in *AMLTA*. Springer, 2012.
[30] http://logic.mimuw.edu.pl/~rses/, Rough Set Exploration System.
[31] http://rsproject.mimuw.edu.pl/, Rseslib is a library of machine learning data structures and algorithms implemented in Java.
[32] http://www.cs.waikato.ac.nz/ml/weka/, Weka 3: Data Mining Software in Java.