Conformity analysis with structured query language

INNAR LIIV, REIN KUUSIK, LEO VÕHANDU
Department of Informatics
Tallinn University of Technology
15 Raja Street, 12618 Tallinn
ESTONIA

Abstract: - The paradigm shift from hypothesis-driven to data-driven exploratory data analysis methods has been gaining acceptance in several emerging research areas as self-organization, machine learning, data mining and bioinformatics. In this paper, a compact overview of such data-driven method is given with short discussion about some properties and implementation issues and an implementation with structured query language is presented. One could make use of database capabilities, thereby leveraging on more than a decade of effort spent in making these systems robust, portable, scalable and concurrent.

Key-Words: - conformity analysis, sql, data mining, outlier detection, knowledge discovery from databases

1 Introduction

The method of conformity analysis was initially presented in [1],[2],[3],[4]. The main goal of the analysis is to perform data reorganization for the input in order to align them according to specific property - conformity. In most cases, the rows (will be referred as objects in this paper) in databases are in the arbitrary order of insertion and attributes according to the initial design of the database. Therefore, simple glance at the database will not give much information or insights about the data, its natural organization, behaviour and evolution. However, all the research areas currently imploying classification and clustering, notice several fundamental fallacies of the methods - it is virutally impossible to describe the transformation between the classes and clusters. As the boundaries are usually fuzzy, the observations near the edges of the classes tend to possess a lot of information that would be valuable for the whole model. Conformity analysis aligns the objects and attributes according to nearest-neighbour similarity and therefore establishes a scale of typicality in the data. It is related to the field of outlier detection, which has important applications in the field of fraud detection, network robustness analysis and intrusion detection (review of literature and research issues in outlier detection available at [5]).

In this paper, a compact overview of the method is given (in section 2) and an implementation with structured query language is presented and discussed (in section 3), followed by the conclusion.

2 Conformity analysis

The algorithm for conformity analysis is presented in figure 1.

Objects:
1. Enumeration of the attribute values for each attribute;
2. Replacement of attribute values with the actual frequency of that value within that attribute;
3. Conformity for objects is calculated using the sum of attribute value frequences;

Attributes:
4. Enumeration of the attribute values for each object;
5. Replacement of attribute values with the actual frequency of that value within that object;
6. Conformity for attributes is calculated using the sum of attribute value frequences.

Figure 1. Algorithm for conformity analysis

Let us look at the following numerical example. Table 1 is the initial dataset before any data manipulations and calculations. After calculating the conformity weights for the objects and attributes according to the presented algorithm, we reorder the elements to get the result presented in Table 2. Initially the algorithm has being used for reordering the data table ([1],[2]), but the measure of conformity itself enables to gain insight to the object (customer) behaviour real-time as the data changes and allows the company to tailor their strategies to make the relationship mutually more valuable.

TABLE 1
INITIAL DATASET

	a_1	a_2	a_3	a_4	a_5
O_1	1	0	0	0	0
O_2	0	1	0	1	1
O_3	0	1	0	1	1
O_4	1	1	0	1	0
O_5	0	0	1	0	1
O_6	0	1	1	1	1

TABLE 2
DATASET AFTER CONFORMITY ANALYSIS

	a_2	a_4	a_5	a_3	a_1	***conformity***
O_2	1	1	1	0	0	**20**
O_3	1	1	1	0	0	**20**
O_6	1	1	1	1	0	**18**
O_4	1	1	0	0	1	**16**
O_5	0	0	1	1	0	**14**
O_1	0	0	0	0	1	**12**
conformity	**20**	**20**	**18**	**16**	**12**	

3 Implementation with SQL

If we think about the steps in the algorithm, we can identify mostly enumeration, replacements and sorting. The idea of this paper is to delegate all the calculation steps to the database systems. One could make use of database capabilities, thereby leveraging on more than a decade of effort spent in making these systems robust, portable, scalable and concurrent. Also it is possible to exploit the underlying SQL parallelization.

Table with the initial data in SQL format (same data as the previous examples) is presented in figure 2.

Presented SQL query (in figure 3) is compatible with the following database systems:

- MySQL 4.1.1-alpha-standard;
- Microsoft SQL Server 2000;
- Microsoft Access 2000;
- PostgreSQL Database Server 8.1.0;
- Oracle 10g.

Notable effort was needed for making the query compatible with the listed systems, as the development of the nested subquery functionality has been different for each of the systems.

The results of the query are presented in figure 4. Future work should include several optimizations with the indices and scalability experiments.

```
CREATE TABLE DATA_TABLE ( o int,  a int, v int);

INSERT INTO DATA_TABLE (o,a,v) VALUES ('1', '1', '1');
INSERT INTO DATA_TABLE (o,a,v) VALUES ('1', '2', '0');
INSERT INTO DATA_TABLE (o,a,v) VALUES ('1', '3', '0');
INSERT INTO DATA_TABLE (o,a,v) VALUES ('1', '4', '0');
INSERT INTO DATA_TABLE (o,a,v) VALUES ('1', '5', '0');

INSERT INTO DATA_TABLE (o,a,v) VALUES ('2', '1', '0');
INSERT INTO DATA_TABLE (o,a,v) VALUES ('2', '2', '1');
INSERT INTO DATA_TABLE (o,a,v) VALUES ('2', '3', '0');
INSERT INTO DATA_TABLE (o,a,v) VALUES ('2', '4', '1');
INSERT INTO DATA_TABLE (o,a,v) VALUES ('2', '5', '1');

INSERT INTO DATA_TABLE (o,a,v) VALUES ('3', '1', '0');
INSERT INTO DATA_TABLE (o,a,v) VALUES ('3', '2', '1');
INSERT INTO DATA_TABLE (o,a,v) VALUES ('3', '3', '0');
INSERT INTO DATA_TABLE (o,a,v) VALUES ('3', '4', '1');
INSERT INTO DATA_TABLE (o,a,v) VALUES ('3', '5', '1');

INSERT INTO DATA_TABLE (o,a,v) VALUES ('4', '1', '1');
INSERT INTO DATA_TABLE (o,a,v) VALUES ('4', '2', '1');
INSERT INTO DATA_TABLE (o,a,v) VALUES ('4', '3', '0');
INSERT INTO DATA_TABLE (o,a,v) VALUES ('4', '4', '1');
INSERT INTO DATA_TABLE (o,a,v) VALUES ('4', '5', '0');

INSERT INTO DATA_TABLE (o,a,v) VALUES ('5', '1', '0');
INSERT INTO DATA_TABLE (o,a,v) VALUES ('5', '2', '0');
INSERT INTO DATA_TABLE (o,a,v) VALUES ('5', '3', '1');
INSERT INTO DATA_TABLE (o,a,v) VALUES ('5', '4', '0');
INSERT INTO DATA_TABLE (o,a,v) VALUES ('5', '5', '1');

INSERT INTO DATA_TABLE (o,a,v) VALUES ('6', '1', '0');
INSERT INTO DATA_TABLE (o,a,v) VALUES ('6', '2', '1');
INSERT INTO DATA_TABLE (o,a,v) VALUES ('6', '3', '1');
INSERT INTO DATA_TABLE (o,a,v) VALUES ('6', '4', '1');
INSERT INTO DATA_TABLE (o,a,v) VALUES ('6', '5', '1');
```

Figure 2. Table and the initial data

```
SELECT tmp_o_sum.o, tmp_a_sum.a, DATA_TABLE.v, tmp_o_sum.tmp_o_sum as o_sum, tmp_a_sum.tmp_a_sum as
a_sum
FROM ((SELECT tmp_frequency.o, Sum(tmp_frequency.ver) AS tmp_o_sum
FROM (SELECT DATA_TABLE.o, DATA_TABLE.a, tmp_freq_v.s AS ver, tmp_freq_h.s AS hor
FROM (SELECT o, v, count(*) AS s
FROM DATA_TABLE
GROUP BY o, v) tmp_freq_h INNER JOIN (DATA_TABLE INNER JOIN (SELECT a, v, count(*) AS s
FROM DATA_TABLE
GROUP BY a, v) tmp_freq_v ON (DATA_TABLE.v = tmp_freq_v.v) AND (DATA_TABLE.a = tmp_freq_v.a)) ON
(tmp_freq_h.v = DATA_TABLE.v) AND (tmp_freq_h.o = DATA_TABLE.o)) tmp_frequency
GROUP BY tmp_frequency.o) tmp_o_sum INNER JOIN DATA_TABLE ON tmp_o_sum.o = DATA_TABLE.o) INNER JOIN
(SELECT tmp_frequency.a, Sum(tmp_frequency.hor) AS tmp_a_sum
FROM (SELECT DATA_TABLE.o, DATA_TABLE.a, tmp_freq_v.s AS ver, tmp_freq_h.s AS hor
FROM (SELECT o, v, count(*) AS s
FROM DATA_TABLE
GROUP BY o, v) tmp_freq_h INNER JOIN (DATA_TABLE INNER JOIN (SELECT a, v, count(*) AS s
FROM DATA_TABLE
GROUP BY a, v) tmp_freq_v ON (DATA_TABLE.v = tmp_freq_v.v) AND (DATA_TABLE.a = tmp_freq_v.a)) ON
(tmp_freq_h.v = DATA_TABLE.v) AND (tmp_freq_h.o = DATA_TABLE.o)) tmp_frequency
GROUP BY tmp_frequency.a) tmp_a_sum ON DATA_TABLE.a = tmp_a_sum.a
ORDER BY tmp_o_sum.tmp_o_sum DESC , tmp_a_sum.tmp_a_sum DESC;
```

Figure 3. Conformity analysis with structured query language

o	a	v	o_sum	a_sum
2	2	1	20	20
3	2	1	20	20
2	4	1	20	20
3	4	1	20	20
2	5	1	20	18
3	5	1	20	18
2	3	0	20	16
3	3	0	20	16
2	1	0	20	12
3	1	0	20	12
6	2	1	18	20
6	4	1	18	20
6	5	1	18	18
6	3	1	18	16
6	1	0	18	12
4	2	1	16	20
4	4	1	16	20
4	5	0	16	18
4	3	0	16	16
4	1	1	16	12
5	2	0	14	20
5	4	0	14	20
5	5	1	14	18
5	3	1	14	16
5	1	0	14	12
1	2	0	12	20
1	4	0	12	20
1	5	0	12	18
1	3	0	12	16
1	1	1	12	12

30 rows in set (0.02 sec)

Figure 4. Result of the presented query

4 Conclusion

In this paper, a compact overview of conformity analysis was given and an implementation with structured query language was presented.

It is also possible to define the presented query as a structured query language view, allowing to overlook the general complexity of the query and to develop a conformity view of each dataset. Several industries need the measurement of usual and unusual behaviour in their application and such approach could reduce the time of preprocessing the data and concentrate only on the problem itself.

References:

[1] Võhandu, L., Kuusik, R., Torim, A., Aab, E., Lind, G., "Some algorithms for data table (re)ordering using Monotone Systems," In Proceedings of the 5th WSEAS Int. Conf. on Artificial Intelligence, Knowledge Engineering and Data Bases, pp. 417-422, 2006

[2] Võhandu, L., "Some problems with data analysis," Transactions of Tallinn Technical University, No. 366, pp. 3-14, 1974

[3] Võhandu, L. "Rapid Data Analysis Methods," Transactions of Tallinn Technical University, No. 464, p.21-39, 1979.

[4] Võhandu, L., "Some Methods to Order Objects and Variables in Data Systems," Transactions of Tallinn Technical University, No. 482, pp. 43-50, 1980.

[5] Aggarwal, C.C., Yu, P.S., "Outlier detection for high dimensional data," In Proceedings of the 2001 ACM SIGMOD international conference on Management of data, pp. 37-46, 2001.