Janusz Kacprzyk; Sławomir Zadrożny

On a fuzzy querying and data mining interface

Kybernetika, Vol. 36 (2000), No. 6, [657]--670

Persistent URL: `http://dml.cz/dmlcz/135379`

KYBERNETIKA — VOLUME *36* (2000), NUMBER 6, PAGES 657–670

ON A FUZZY QUERYING AND DATA MINING INTERFACE

JANUSZ KACPRZYK AND SŁAWOMIR ZADROŻNY

In the paper an interface is proposed that combines flexible (fuzzy) querying and data mining functionality. The point of departure is the fuzzy querying interface designed and implemented previously by the present authors. It makes it possible to formulate and execute, against a traditional (crisp) database, queries containing imprecisely specified conditions. Here we discuss possibilities to extend it with some data mining features. More specifically, linguistic summarization of data (databases), as introduced by Yager [16], is advocated as an interesting extension of simple querying. The link between linguistic (fuzzy) data summaries and association rules is discussed and exploited.

1. INTRODUCTION

A fuzzy querying interface, as meant here, makes it possible for a user to employ his or her own dictionary of linguistic terms to be used in queries. These linguistic terms provide for a direct representation of presumed vagueness and imprecision of queries. For example, a customer of a real-estate agency looking for a house would rather express his or her criteria using imprecise descriptions as cheap, large garden, etc. Also, to specify which combination of the criteria fulfillment would be satisfactory, he or she would often use expressions like *most of them* or *almost all* (should be fulfilled). The semantics for such linguistic terms is provided by fuzzy logic. This idea has been implemented in the whole family of fuzzy querying interfaces, notably the FQUERY for Access package proposed and implemented by the authors [6].

The possibility to use fuzzy linguistic terms should be attractive not only for a non-expert, casual user. An implicit aggregation power of the proposed querying formalism may also be useful for some decision making related databases inquiries. This may be made even more attractive as the same linguistic paradigm and user interface may be applied for the purposes of data mining. Linguistic, imprecise terms used to represent rules, patterns etc. may much more easily account for noisy, incomplete etc. data. Moreover, using a linguistic representation for pieces of discovered knowledge, we can immediately present it to the end-user, without any additional representation transformations. For example, a database summary like "most our customers are reliable" may be more useful than, say, "65 % of our customers have paid at least 70 % of their duties in less than 10 days". From this perspective, fuzzy (linguistic) queries are directly applicable for the data mining purposes. Both approaches may be regarded as dealing with the same building blocks. Moreover, the data mining efforts may be guided by a recorded history of fuzzy queries posed against a given database.

The structure of the paper is as follows. First, we briefly sum up Zadeh's calculus of linguistically quantified propositions that is a formal foundation for our approach to fuzzy querying as well as database summaries. Then, we briefly introduce Yager's concept of linguistic summaries of databases extending it towards a richer structure. We introduce a classification of database summaries based mainly on their complexity. In the next section, we propose to adapt well-known association rules mining algorithms as a viable solution for mining more complex database summaries. Finally, we present how the mining of linguistic summaries may be embedded within our fuzzy querying interface. We conclude the paper with ideas for further research.

2. CALCULUS OF LINGUISTICALLY QUANTIFIED PROPOSITIONS

A calculus of linguistically quantified propositions, as introduced by Zadeh [17, 18], makes it possible to calculate the truth-value of a proposition of the following type:

$$Qx\text{'s are } S \tag{1}$$

where Q denotes a *fuzzy linguistic quantifier* (e.g., "*most*"), $X = \{x_1, \ldots, x_n\}$ is a universe of discourse, and $S(.)$ is a property (predicate) which is assumed fuzzy and its interpretation may be informally equated with a fuzzy set, i.e.:

$$\text{truth}(S(x_i)) = \mu_S(x_i).$$

In the context of database querying, such a linguistically quantified proposition may be exemplified by:

$$\text{"Most } (Q) \text{ rows } (x\text{'s}) \text{ match the query } (S)\text{".} \tag{2}$$

The scope of the quantifier F may be added yielding

$$QFx\text{'s are } S \tag{3}$$

exemplified by:

$$\text{"Most } (Q) \text{ rows } (x\text{'s}) \text{ passing filter } (F) \text{ match the query } (S)\text{".} \tag{4}$$

Here, F may be treated as another fuzzy property, i.e.:

$$\text{truth}(F(x_i)) = \mu_F(x_i).$$

Basically, Zadeh's calculus consists in devising a way to find truth(Qx's are S) in case of (1) or truth(QFx's are S) in case of (3).

As mentioned above, properties S and F are assumed to be fuzzy and equated with fuzzy sets in X. Linguistic quantifier Q is also represented by a fuzzy set, this time in $[0, 1]$ as, e. g.,

$$\mu_{\text{"}most\text{"}}(y) = \begin{cases} 1 & \text{for } y \geq 0.8 \\ 2y - 0.6 & \text{for } 0.3 < y < 0.8 \\ 0 & \text{for } y \leq 0.3. \end{cases} \tag{5}$$

Then

$$\text{truth}(Qx\text{'s are } S) = \mu_Q\left(\textstyle\sum\text{Count}(S) \big/ \sum\text{Count}(X)\right) = \mu_Q\left(\frac{1}{n}\sum_{i=1}^{n}\mu_S(x_i)\right) \tag{6}$$

and

$$\begin{aligned} \text{truth}(QFx\text{'s are } S) &= \mu_Q\left(\textstyle\sum\text{Count}(F \text{ and } S) \big/ \sum\text{Count}(F)\right) = \\ &= \mu_Q\left(\sum_{i=1}^{n}(\mu_F(x_i) \wedge \mu_S(x_i)\right) \Big/ \sum_{i=1}^{n}\mu_F(x_i)) \end{aligned} \tag{7}$$

where "$\wedge$" is the minimum operation, i. e., $a \wedge b = \min(a, b)$, that, in general, can be replaced by another t-norm, and $\sum$Count is (nonfuzzy) cardinality of a fuzzy set.

3. LINGUISTIC SUMMARIES OF A DATABASE CONTENT

Data mining, also known as *knowledge discovery in databases*, is concerned with the seeking for interesting patterns, dependencies, regularities etc. in bodies of data stored in databases. A database represents a part of the real word under consideration as, e. g., customers, employees, orders etc. of a company, and their relationships. Here we consider the relational database model. Then, the primary component of a database is a *table* that may represent both particular objects of the modelled reality and relationships between them. In the former case, the table collects information on the whole class of objects of the same type (e. g., customers), characterized by a set of *attributes* (*columns*). Each *row* of the table describes one particular object from a given class. One of basic services offered by a *database management system* (DBMS) is to find objects from a given class that meet some conditions. Such conditions taken together form a *query* that is processed by the DBMS giving as the result a set of rows (objects).

There are many formalisms used to construct queries, but the most popular is without a doubt the *Structured Query Language*, SQL. SQL, supported by most of the commercially used DBMS's, offers a rich functionality solving most of typical business-related data processing tasks. Still, there are many enhancements as well as completely new concepts around. For example, the object-oriented paradigm and its associated query languages promise a better representation and processing of more complex data structures. For the topic of this paper two other arguable limitations of classical query languages, including SQL, are important.

Firstly, a precise specification of conditions of a query is not always possible. The use of linguistic terms, imprecise or vague, is often postulated. For example, the user may be interested in retrieving some data concerning "*young* employees". While the meaning of "*young*" is usually obvious for a human being, it may be difficult to express it using a precise criterion of the form "*age* < *?*". Whatever number we put in place of "?", the criterion will still be not satisfactory. Moreover, it is argued that the concepts such as "*young*" require to abandon binary logic: an employee may be not only definitely young or not young, but also maybe young *to some extent, to a certain degree.* There are some approaches based on the application of fuzzy logic that address this problem. The concepts behind that as well as a pilot implementation of the family of FQUERY packages proposed by the authors [6, 7] belongs to them.

Secondly, sometimes the user would like to obtain some information on general characteristics of the objects as, e. g., employees, whose description is stored in a database. In such a case we might say that the user is looking for a *summary* of data. The traditional query languages do not support well such queries as, e. g., "What are predominant features of our employees?". Here, the user does not compose a query but the situation is to some extent opposite. Namely, we expect some queries *matched by some majority of rows (objects)* will be automatically constructed and presented to us. This idea has been developed and studied by Yager [16].

Yager proposed to express a summary of the database using linguistically quantified propositions, as defined by (1) and/or (3). The former state properties of the whole class of objects under consideration, e. g., "*most employees are young*". The latter correspond to IF–THEN rules (i. e. have a conditional character) as, e. g., "*most young employees earn around 30,000 USD*" what may be interpreted also as "IF *an employee is young* THEN *he or she earns around 30,000 USD*". The truth-value of a given linguistically quantified proposition, calculated as in (7), is a primary measure of quality of a given summary. Practically, such a basic quality indicator has to be supplemented with some measure of "interestingness" ("non-triviality", "unexpectedness",...), see, e. g., [4, 5]. In his original approach Yager does not envisage any specific algorithm to find "good" summaries.

In this paper we discuss the question of finding summaries in a more detailed way. In order to do that we will analyse the structure of such a summary on an example. Let us assume a database containing information on some companies. Each company is described with a set of attributes, including *Number of employees*, *Volume of export* and *Profit.* Then, we may consider the structure of the following, exemplary, summary (corresponding to a linguistically quantified proposition, as defined by (3)):

"*Most* companies employing *small number of workers* and *exporting a lot* are *highly profitable*". (8)

We can distinguish the following components of this summary (referring to the filter F and query S mentioned in (3)):

Skeleton of the filter, i. e. the attributes that are constrained by the conditions appearing in the filter (here: *Number of employees*, *Volume of export*) together with logical connectives (here: *and*).

Skeleton of the query, i. e. the attributes that are constrained by the conditions appearing in the query (here: *Profit*) together with logical connectives (here not used).

Fuzzy values defining actual fuzzy constraints put on the attributes in the filter (here: *small*, *a lot*) and the query (here: *highly*).

Linguistic quantifier representing the fraction of rows (companies) matching filter F that at the same time match the query S (here: *most*).

Now, let us consider a hypothetical subsystem of a database management system that will support the generation of summaries. Basically, we can leave for this subsystem all decisions as to the structure of the summaries sought. In a more practical scenario, the user will indicate which table(s) and/or views are of interest. Anyway, the search space includes here the skeletons of the filter and the query as well as particular fuzzy values and a linguistic quantifier to be used. In such a case the process of data summarization will be extremely time-consuming but may produce really interesting results. Some approaches along this direction employed genetic algorithm [3].

In order to limit the computational complexity of the problem, we may start with a template of the summary, specifying most of its components and asking the system to find the remaining ones. Again, in the extreme scenario, the user guesses a candidate summary and the system only evaluates its quality.

There are many other scenarios, that locate themselves somewhere between these two extreme cases, using more or less completely specified template of the sought summaries. Let us consider the particular cases corresponding to the earlier identified components of the summary. The simplest case concerns the linguistic quantifier – it may be specified in the template or left out to be found by the system. In case of filter F and query S, more scenarios may be considered. Thus, depending on the form of the template, we can introduce a certain classification of the summaries. This classification is presented in Table 1. The components specified in the template as well as sought are denoted in the following way:

- F (S) – the skeleton as well as embedded fuzzy values of the filter (query),
- F^{fc} (S^{fc}) – the skeleton of the filter (query),
- F^{v} (S^{v}) – fuzzy values to appear in the filter (query)
- Q – linguistic quantifier.

Summaries of type 1 and 3 refer to linguistically quantified proposition defined by 1, while the other types are based on Definition 3.

The type 1 summary may be easily produced by a simple extension of fuzzy querying as proposed and implemented in our FQUERY for Access package, see the next section for more details. Basically, the user has to construct a query – a

Table 1. Classification of summaries.

Type	Given	Sought	Remarks
1	S	Q	Simple summaries through ad-hoc queries
2	S	F, Q	Conditional summaries through ad-hoc queries
3	Q, S^{fc}	S^v	Simple value oriented summaries
4	Q, S^{fc}, F	S^v	Conditional value oriented summaries
5	nothing	$S, F, Q,$	General fuzzy rules

candidate summary. Then, it has to be determined what the fraction of the rows matching this query is and what linguistic quantifier best denotes this fraction. The primary target of this type of summarization is certainly to propose such a query that a large proportion, e. g., most, of the rows satisfies it. On the other hand, it may be interesting to learn that only few rows satisfy some meaningful query. The type 2 summary is an extension of a type 1 summary by addition of a fuzzy filter. For more on these types of summaries, see for instance [8], and [2] for a non-fuzzy approach.

The summary of type 3 is computationally much more difficult to be found. Its most restricted (and most practical) version may be interpreted as expressing the typical value of an attribute. In such a special case, query S consists of only one simple condition built of the attribute whose typical value is sought. For example, the template for such a query may look like follows: "*Most customers are of* $\langle$*fuzzy value*$\rangle$ *size*". Here the system has to determine what fuzzy value (e. g., *large, middle, small*), if any, makes this summary valid, i. e., true to high enough degree. If there is such a fuzzy value, it may be interpreted as typical value for the attribute *Size* of our customers. This type of summaries may be used with more complicated, regular queries but it may quickly become computationally infeasible (due to the combinatorial explosion) and the interpretation of results becomes vague. The type 4 summary may produce typical value of selected attribute for some, possibly fuzzy, subset of rows.

The type 5 summary is the most comprehensive form considered here. Namely, we do not assume anything about the structure of the summary sought. Thus, in general what we receive is a set of IF–THEN rules. For a general form of such a rule it is difficult to devise an effective and efficient generation algorithm, as mentioned earlier. In the next section, we discuss a special case of Type 5 summary, for which computationally efficient algorithms are known in the literature.

4. LINGUISTIC ASSOCIATION RULES

The classification of summaries presented in the previous section clearly indicates the type 5 summaries as the most interesting but most difficult to find. Here, we try to explore a subset of type 5 summaries exploiting their similarity to what is known in the literature as *association rules* [1].

Originally, the association rules were defined for binary valued attributes in the following form:

$$A_1 \wedge A_2 \wedge \ldots \wedge A_n \to A_{n+1}. \tag{9}$$

Such an association rule states that if for a database row all the attributes from the set $\{A_1, A_2, \ldots, A_n\}$ take on value 1, then also the attribute A_{n+1} takes on value 1. A row in a database is said to support a set of attributes $\{A_i\}_{i \in I}$ if all attributes from the set take on in this row value 1. This form of association rules was motivated by their early applications for so-called customer's basket analysis.

There are two measures of the quality of an association rule used:

- the support of a rule (9) is the fraction of the rows supporting the set of attributes $\{A_i\}$, $i \in \{1, \ldots, n+1\}$, and
- the confidence of a rule is the fraction of the rows supporting the set of attributes $\{A_i\}, i \in \{1, \ldots, n+1\}$ among all rows supporting the set of attributes $\{A_i\}, i \in \{1, \ldots, n\}$.

Thus, while the support determines a statistical significance of a rule, the confidence measures its strength in the database. Usually, we are interested in rules having values of the support measure above some minimal threshold and a high value of the confidence measure. Setting the threshold in a reasonable way, we can essentially reduce the size of the space of possible association rules and at the same time not ignore any interesting rule.

A number of efficient algorithms for finding all association rules possessing a required support measure were devised, see, e. g. [1, 12]. The original concept of the association rule essentially evolved over time but still the same algorithms are applicable. The extensions to the initial form of the association rule include:

I. the right-hand side of (9) containing a conjunction of the attributes instead of just one attribute,

II. so-called generalized association rules where cardinality of attributes' domains may be grater than 2 and a hierarchies may be imposed on domains; [13],

III. numerical, real-valued attributes may be used in rules [14],

IV. some constraints may be imposed on combinations of attributes used in rules [15].

In the context of database summaries the most important is the extension III. Basically, the idea is to discretize the continuous domain and to treat every discrete value as a new, artificial binary attribute. This way, it is possible to employ standard association rules mining algorithm for binary attributes. Thanks to this extension, we can interpret the association rules as a subset of our type 5 summaries. This may be verified by the following observations. The left-hand and right-hand sides of the association rule (9) correspond to the filter F and the query S of the summary, respectively. The structure of such a filter and query is rather limited (just a conjunction), but this simplicity secures the existence of efficient algorithms for rules generation. The truth-value of the summary, defined by (7), corresponds to the confidence measure of a rule.

The applicability of the linguistic terms (fuzzy values) for the construction of the association rules was quickly observed, see, e. g. [10]. The concept of the querying employing linguistic terms, as implemented in our FQUERY for Access package, offers an interesting framework for the association rules mining. For example, the discretization of the continuous domain is readily available. Namely, the domain of an attribute is covered by linguistic terms, that are used for querying purposes. A fuzzy querying interface makes it even simpler as a dictionary of linguistic terms is maintained for the user. Moreover, these linguistic terms are familiar for the user as they were defined and tested by him or her in some fuzzy queries. Thus, the meaning of rules produced using such linguistic terms (fuzzy values) should be much more clear to the user.

The suitability of a fuzzy querying interface for association rules mining purposes is verifiable also from the software engineering perspective. In the next section we briefly present how it can be organized.

5. FUZZY QUERYING AND DATA MINING VIA FQUERY FOR ACCESS

Basically, the fuzzy querying, as meant here, consists in a use of linguistic terms in queries. The meaning of linguistic terms is often vague. Thus, some special formal means have to be adopted to deal with them. Moreover, it would be unreasonable to require the answer for a vague query to be completely precise, following the classical yes-no logic. Instead of a crisp set of matching database rows, a fuzzy query yields a fuzzy set of such rows. Namely, each row is accompanied by its matching degree against the query. We employ fuzzy logic to represent the meaning of the linguistic terms used in queries. The following types of linguistic terms are considered:

- **numerical fuzzy values**, exemplified by *young* in "young employee",
- **non-numerical fuzzy values**, exemplified by *Central Europe* in "country is in Central Europe",
- **fuzzy relations**, exemplified by *much greater than* in "export is much greater than import", and
- **linguistic quantifiers**, exemplified by *most* in "most conditions have to be met".

All these linguistic terms, are represented as fuzzy sets. Numerical fuzzy values may be used in a query along with a numerical attributes and are defined as trapezoidal fuzzy numbers in the domain of this attribute. In our approach, we make it possible to define the universal, context-independent numerical fuzzy values; for more details see, e. g., [6]. Non-numerical fuzzy values are appropriate for qualitative attributes. For example, an imprecise term "Central Europe" is meaningful only in the context of attributes related to the country.

Classically, in a query particular conditions may be combined using the classical logical connectives AND and OR. In our approach, linguistic quantifiers provide for a more flexible aggregation scheme of conditions in queries. For example, instead of requiring that all simple conditions are met, using an appropriate linguistic quantifier one may indicate that most of them are to be met. The linguistic quantifiers are interpreted as in Zadeh's calculus of linguistically quantified propositions, see Section 2.

The definition of a linguistic term consists of a label and a membership function of the associated fuzzy set. Definitions of all linguistic terms known to the system, including these defined by users as well as predefined in the system, are maintained in the dictionary. This feature supports also data mining functionality, to be discussed next. The concept of fuzzy querying, briefly presented here, has been implemented in our software, the FQUERY for Access package. It is an add-in to Microsoft Access providing the user of this popular desktop database management system with fuzzy querying capabilities.

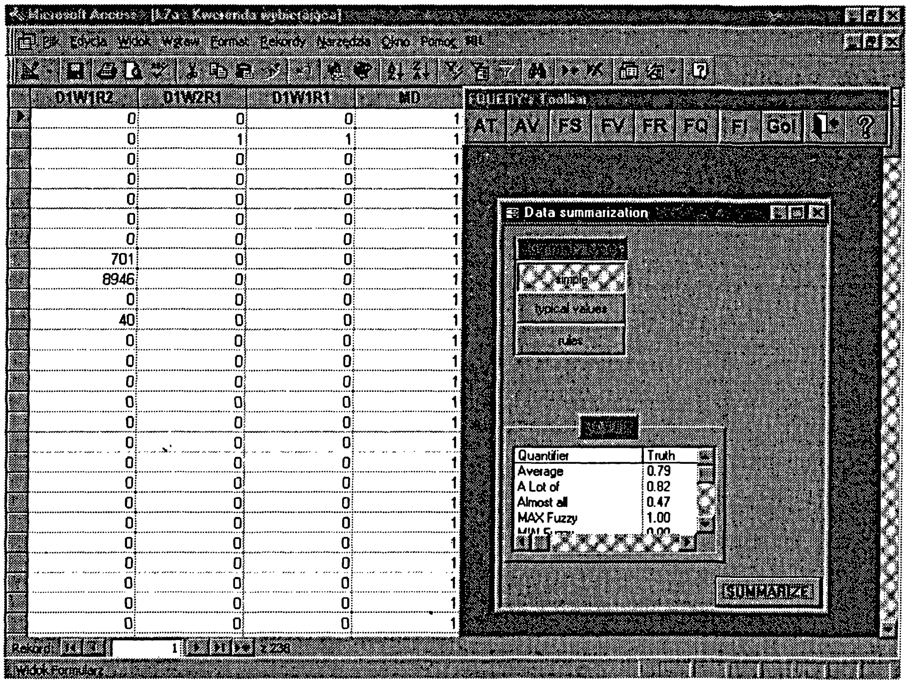
The fuzzy querying interface is itself an interesting tool for data mining. Ad-hoc created queries, one of the simplest data mining techniques, become much more powerful when linguistic terms are supported. Nevertheless, the implementation of various types of summaries mentioned in Section 3 seems to be worthwhile. The dictionary of linguistic terms plays here crucial role. These linguistic terms are main building blocks of summaries. We need only an efficient, compatible with the rest of the fuzzy querying system, procedure to find summaries. The type 1 summary (referred to also as *simple* summary) require determining a linguistic quantifier best describing the fraction of rows satisfying query S. Hence, we are looking for a fuzzy set A in the space of linguistic quantifiers, such that:

$$\mu_A(Q) = \text{truth}(Qx\text{'s are } S) = \mu_Q\left(\sum_{i=1}^{n} \mu_S(x_i)/n\right) \tag{10}$$

where $X = \{x_1, \ldots, x_n\}$ is the set of rows.

FQUERY for Access processes the query in the usual way calculating the degree of matching for each row. Additionally, all matching degrees are summed up yielding the sum in (10). Then the results of the query, i. e. rows accompanied by a matching degree, are displayed in the usual form. In another window, the sought fuzzy set of linguistic quantifiers is shown, as shown in Figure 1. Currently, FQUERY for Access does not support fuzzy filters. As soon as this capability is added, also summaries of type 2 will be available.

In case of a type 3 summary (referred to also as *typical values* summary), the quantifier and the structure of the query are given, but some conditions of the query are not fully specified. Let us consider a simple example, where the query has the form "*export is ?*". Thus, the query is not fully specified and the system has to check with which linguistic term to replace the question mark so as to receive highest truth



D1W1R2	D1W2R1	D1W1R1	MD
0	0	0	1
0	1	1	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
701	0	0	1
8946	0	0	1
0	0	0	1
40	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1
0	0	0	1

Quantifier	Truth
Average	0.79
A Lot of	0.82
Almost all	0.47
MAX Fuzzy	1.00
MIN F	0.0

Fig. 1. An example of Type 1 summary.

value of resulting linguistically quantified proposition. Hence, in this case we are looking for a fuzzy set B in the space of linguistic terms (numerical fuzzy values) $\{V_i\}$, such that:

$$\mu_B(V_i) = \text{truth}(Qx\ x.export = V_i) \tag{11}$$

where $x.export$ denotes the value of attribute *export* for row x.

During the query processing the question marks are treated in the same way as regular numerical fuzzy values. However, the matching degree is calculated not just for one, concrete fuzzy value, but for all fuzzy values defined in the dictionary. The matching degrees of the whole query against the subsequent rows, calculated for different combination of fuzzy values are summed up. Finally, it is computed for particular combinations of fuzzy values how well the query is satisfied when a given combination is put into the query. Obviously, such computations are extremely time-consuming and are practically feasible only for one question mark used in a query.

Finally, we will briefly discuss how we implement a subset of type 5 summaries using association rules mining algorithms. Our implementation of association rules is based on Agrawal and Srikant's AprioriTid algorithm [1]. Basically, the original algorithm (for binary data) consists in finding the *large itemsets*, i. e. the itemsets for which the support measure (see Section 4) is greater than certain, pre-defined threshold value. An itemset containing k items is called a k-itemset.

In our implementation we deal with real-valued attributes. Thus, following previously mentioned approaches, for each such an attribute we introduce a number of artificial attributes (items) which may be treated as binary attributes. For example, the attribute PRICE may be replaced with the following artificial attributes: PRICEIsLow, PRICEIsMedium and PRICEIsHigh. The meaning of these attributes is obvious: a row supports, for example, PRICEIsLow, if the value of the original attribute PRICE in this row falls into interval identified as 'Low'. Instead of the intervals of values, employed in non-fuzzy approaches, we use fuzzy values (linguistic terms). The use of fuzzy values implies that a row supports given artificial attribute to a degree from [0,1]. It is possible to use such a degree directly or to "defuzzify" it employing a threshold.

The current implementation of the associations rules mining algorithm in our FQUERY for Access package consists of the following steps.

Step 1. Selection of attributes and linguistic terms (fuzzy values) to be used in association rules sought. The use of all attributes defined in the database (table or view) may be impractical for two reasons. First of all, it requires a lot of computations. Secondly, some attributes may be a priori judged as non-interesting components of the rules sought. The former reason applies also to the linguistic terms (fuzzy values).

Step 2. For each pair (attribute,fuzzy value) new, artificial item is constructed, as it is described earlier. All items constitute the set of 1-itemsets, LIS.

Step 3. The support for each 1-itemset is calculated. Here, the fuzzy querying module is employed, as it is required to compute the matching degree of each item (containing linguistic terms) against every row in database (table, view). Thus, as a result we obtain a fuzzy set of rows supporting given item – the membership degree equals the computed matching degree. Then, the *the fuzzy support measure* of the item is the $\sum$Count of this fuzzy set, while the standard, "crisp" support measure corresponds to the classical cardinality of the α-cut of this fuzzy set (α corresponds to one of the parameters of the algorithm – the threshold of matching). Optionally, this or other support measure may be employed in the next steps of the algorithm.

Step 4. The items with the support below the *threshold1* and above the *threshold2* are removed from the set LIS. The former condition is standard for the association rules mining algorithm. The second condition is added here to avoid the items that are present in almost all rows (this threshold is set 80 – 90 %).

Step 5. The calculation of the support for 1-itemsets requires the matching degree computation and is rather expensive. As this has to be repeated for k-itemsets, $k > 1$, we adopt here the idea of AprioriTid algorithm. Thus, the transaction table indicating for each row, which 1-itemsets are supported by this row to the degree higher than matching threshold, is created. This trick trades CPU complexity for disk storage requirements.

Set $k = 2$

Step 6. The k-itemsets are constructed, i. e., itemsets containing k items. This is done as in the original AprioriTid algorithm.

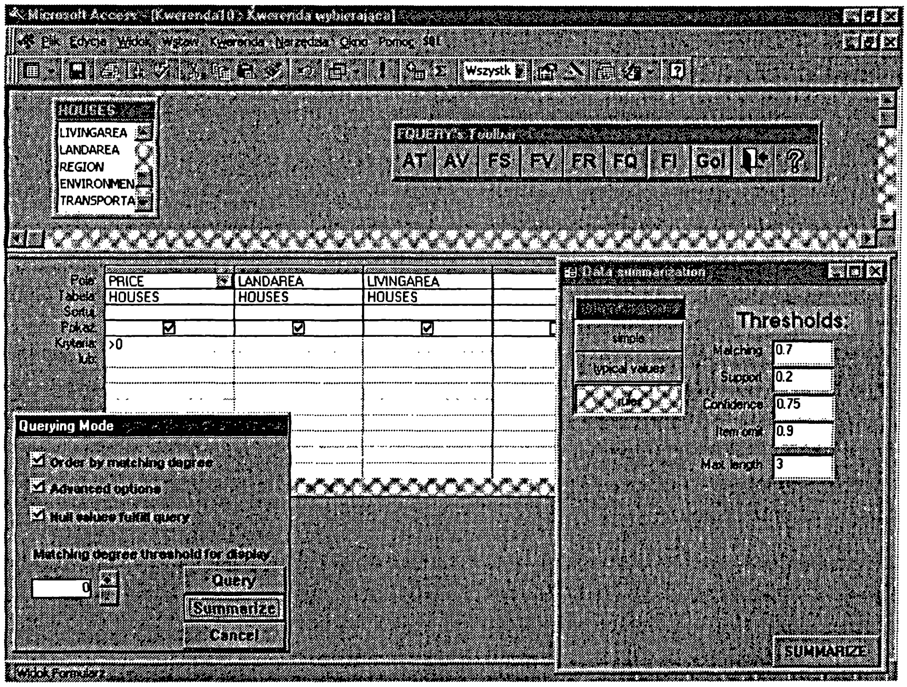
Step 7. The support for k-itemsets is calculated. This is done as in the Step 3, but this time the transaction table generated for $(k-1)$-itemsets is employed.

Step 8. These k-itemsets that are supported to the high enough degree (above *threshold1*, are added to the set *LIS*.
IF no k-itemsets were added to the set *LIS* THEN GOTO Step 10

Step 9. Similarly as in Step 5, the transaction table is constructed but this time for k-itemsets.
SET $k = k + 1$; GOTO Step 6

Step 10. Generate rules from large l-itemsets collected in the set *LIS*. This is done as in the standard algorithm.
STOP.

The procedure of generating association rules in FQUERY for Access is illustrated in Figure 2.



Pole:	PRICE	LANDAREA	LIVINGAREA
Tabela:	HOUSES	HOUSES	HOUSES
Sortuj:			
Pokaż:	☑	☑	☑
Kryteria:	>0		
lub:			

☑ Order by matching degree
☑ Advanced options
☑ Null values fulfill query
Matching degree threshold for display: 0

Thresholds:

Matching	0.7
Support	0.2
Confidence	0.75
Item omit	0.9
Max length	3

Fig. 2. Setting parameters for Type 5 summaries (association rules) mining.

6. CONCLUDING REMARKS

We have advanced the concept of linguistic summarization of the database content. The proposed classification of summaries takes into account their complexity. We conclude that in case of some types of summaries it is fairly easy to find them. In some other cases, especially type 5 summaries, it is generally very difficult. We consider the possibility to use well-known association rules mining algorithms to find a subset of type 5 summaries. In order to do that, we analyse the similarity of both concepts. Finally, we discuss how summaries mining may be combined with the flexible (fuzzy) querying. We briefly present a pilot implementation done within our FQUERY for Access package.

Further research on efficient algorithms for *linguistic association rules* may be fruitful. We are going to explore also the connections of the association rules and classifiers as pointed out by some authors [11].

(Received June 13, 2000.)

REFERENCES

[1] R. Agrawal and R. Srikant: Fast algorithms for mining association rules. In: Proc. 20th Internat. Conference on Very Large Databases, Santiago 1994.

[2] T. M. Anwar, H. W. Beck and S. B. Navathe: Knowledge mining by imprecise querying: A classification based system. In: Proc. Internat. Conference on Data Engineering, Tampa, USA 1992, pp. 622–630.

[3] R. George and R. Srikanth: Data summarization using genetic algorithms and fuzzy logic. In: Genetic Algorithms and Soft Computing (F. Herrera and J. L. Verdegay, eds.), Physica–Verlag, Heidelberg – New York 1996, pp. 599–611.

[4] J. Kacprzyk and P. Strykowski: Linguistic data summaries for intelligent decision support. In: Fuzzy Decision Analysis and Recognition Technology for Management, Planning and Optimization – Proceedings of EFDAN'99 (R. Felix, ed.), Germany 1999, pp. 3–12.

[5] J. Kacprzyk and P. Strykowski: Linguistic Summaries of Sales Data at a Computer Retailer: A Case Study. In: Proceedings of IFSA'99 (Taipei, Taiwan R.O.C), vol. 1, 1999, pp. 29–33.

[6] J. Kacprzyk and S. Zadrożny: FQUERY for Access: fuzzy querying for a Windows-based DBMS. In: Fuzziness in Database Management Systems (P. Bosc and J. Kacprzyk, eds.), Physica–Verlag, Heidelberg 1995, pp. 415–433.

[7] J. Kacprzyk and S. Zadrożny: Flexible querying using fuzzy logic: An implementation for Microsoft Access. In: Flexible Query Answering Systems (T. Andreasen, H. Christiansen and H. L. Larsen, eds.), Kluwer, Boston 1997, pp. 247–275.

[8] J. Kacprzyk and S. Zadrożny: Data mining via linguistic summaries of data: An interactive approach. In: Methodologies for the Conception, Design and Application of Soft Computing (T. Yamakawa and G. Matsumoto, eds., Proceedings of IIZUKA'98), Iizuka 1998, pp. 668–671.

[9] J. Kacprzyk and S. Zadrożny: On summarization of large datasets via a fuzzy-logic-based querying add-on to Microsoft Access. In: Intelligent Information Systems VII, Malbork, IPI PAN, Warsaw 1998, pp. 249–258.

[10] J.-H. Lee and H. Lee-Kwang: An extension of association rules using fuzzy sets. In: Proc. Seventh IFSA World Congress, Prague 1997, Vol. 1, pp. 399–402.

[11] B. Liu, W. Hsu and M. Yiming: Integrating Classification and Association Rule Mining. In: Proc. Fourth Internat. Conference on Knowledge Discovery and Data Mining (KDD-98, Plenary Presentation), New York 1998.

[12] H. Mannila, H. Toivonen and A. I. Verkamo: Efficient algorithms for discovering association rules. In: Proc. AAAI Workshop on Knowledge Discovery in Databases (U. M. Fayyad and R. Uthurusamy, eds.), Seattle 1994, pp. 181–192.

[13] R. Srikant and R. Agrawal: Mining generalized association rules. In: Proc. 21st Internat. Conference on Very Large Databases, Zurich 1995.

[14] R. Srikant and R. Agrawal: Mining quantitative association rules in large relational tables. In: Proc. ACM–SIGMOD 1996 Conference on Management of Data, Montreal 1996.

[15] R. Srikant, Q. Vu and R. Agrawal: Mining association rules with item constraints. In: Proc. 3rd Internat. Conference on Knowledge Discovery in Databases and Data Mining, Newport Beach 1997.

[16] R. R. Yager: On linguistic summaries of data. In: Knowledge Discovery in Databases (G. Piatetsky–Shapiro and W. J. Frawley, eds.), AAAI Press/The MIT Press, Menlo Park 1991, pp. 347–363.

[17] L. A. Zadeh: A computational approach to fuzzy quantifiers in natural languages. Comput. Math. Appl. *9* (1983), 149–184.

[18] L. A. Zadeh: A computational theory of dispositions. Internat. J. Intelligent Systems *2* (1987), 39–64.

Prof. Dr. Janusz Kacprzyk and Dr. Sławomir Zadrożny, Systems Research Institute, Polish Academy of Sciences and University of Information Technology and Management, ul. Newelska 6, 01-447 Warszawa. Poland.
e-mails: kacprzyk, zadrozny@ibspan.waw.pl