Data Cleaning and Semantic Improvement in Biological Databases

Daniele Apiletti, Giulia Bruno, Elisa Ficarra, Elena Baralis

Dep. of Control and Computer Engineering (DAUIN),
Politecnico di Torino, C.so Duca degli Abruzzi 24, 10129 Torino, Italy

Summary

Public genomic and proteomic databases can be affected by a variety of errors. These errors may involve either the description or the meaning of data (namely, syntactic or semantic errors). We focus our analysis on the detection of semantic errors, in order to verify the accuracy of the stored information. In particular, we address the issue of data constraints and functional dependencies among attributes in a given relational database. Constraints and dependencies show semantics among attributes in a database schema and their knowledge may be exploited to improve data quality and integration in database design, and to perform query optimization and dimensional reduction.

We propose a method to discover data constraints and functional dependencies by means of association rule mining. Association rules are extracted among attribute values and allow us to find causality relationships among them. Then, by analyzing the support and confidence of each rule, (probabilistic) data constraints and functional dependencies may be detected. With our method we can both show the presence of erroneous data and highlight novel semantic information. Moreover, our method is database-independent because it infers rules from data.

In this paper, we report the application of our techniques to the SCOP (Structural Classification of Proteins) and CATH Protein Structure Classification databases.

1 Introduction

It's about thirty years that biological data are generated from a variety of biomedical devices and stored at an increasing rate in public repositories. Recently a big effort has been made to integrate distributed heterogeneous databases, where researchers continuously store their new experimental results. However data quality improvement is one of the foremost tasks to perform, since the accuracy of data analysis and the ability to produce correct results from data mining relies on it.

Public biological databases can be affected by a variety of errors, which may involve either the description or the meaning of information. The existence of such erroneous or poor data harmfully affects any further elaboration or application.

Typically addressed data quality problems can be divided into two categories: syntactic anomalies and semantic anomalies. Among syntactic anomalies there are the problems of *incompleteness* (due to the lack of attribute values), *inaccuracy* (due to the presence of errors and outliers), *lexical* errors, *domain format* errors and *irregularity*. Among semantic anomalies there are *discrepancy*, due to a conflict between some attribute values (i.e. age and date of birth), *ambiguity*, due to the presence of synonyms, homonyms or abbreviations, *redundancy*, due to the presence of duplicate information, *inconsistency*, due to an integrity constraint violation (i.e. the attribute age must be a value grater than 0) or a functional constraint violation (i.e. if the attribute married is false, the attribute wife must be null),

invalidity, due to the presence of tuples that do not display anomalies of the classes defined above but still do not represent valid entities [1].

In data mining terminology, the deletion of these anomalies is known as *Data Cleaning*. To get quality mining results it is important to remove these problems before analysing the data. Public genomic and proteomic databases could be affected by the afore-mentioned anomalies, mainly because they grew some years ago, under the pressing need of storing the large amount of genetic information available at the time, without having a standard method to collect it neither a standard format to store it. As data acquisition techniques improve and the sources of data differentiate, the formats of genetic information become heterogeneous getting harder to maintain and improve the quality of such data.

Errors can occur in different phases of data production: experiment (unnoticed experimental setup failure or systematic errors), analysis (misinterpretation of information), transformation (from one representation into another), propagation (erroneous data used for the generation of new data), and staleness (unnoticed changes in data could produce the falsification of other data which depend on it). These problems lead to semantic errors and the resulting information does not represent the real-world facts correctly. Data dependencies inherent to the production process and to the usage of the data make genome data predestined for propagated errors [2].

For these reasons, there is the need of a data cleaning framework, able to automatically recognize the quality problems in databases and to address them to domain experts for further investigation.

In the last few years, the need of data cleaning tools has increased, and some methods have been proposed to obtain quality improvement. Some efforts have been made to solve the problem of duplicated records (sometimes called instance identification, object identity or record linkage problem) [3], [4]. This case is, for instance, when the same entities do not have identical labels, e.g. “gene_321” and “g_321”. This is a common problem also for the integration of bibliographical and text information written in different formats. In [5] a generic knowledge-based framework for data cleaning is proposed, but it can only identify duplicates and anomalies, even if with high recall and precision.

Most existing works focus on inaccuracy, lexical errors, redundancy problems and enforcement of integrity constraints, but ignore the functional constraint violations.

Since quality problems depend on the particular database considered, it is not advisable trying to define a-priori rules to check data for errors, because they would change from a database to another. Moreover, due to the large amount of data in existing databases, it is difficult, also for a specialist, to recognize the structure and the relationships among attributes. Instead it would be better to define an algorithm that automatically infers rules from data: they can change, but the algorithm remains the same.

We propose a method to discover constraints and functional dependencies among data by means of association rule mining. Constraints and dependencies show semantics among attributes in a database schema. Their knowledge can be exploited to improve data quality and integration in database design, and to perform query optimization and dimensional reduction. Association rules are a well-known data mining tool. They have been applied to biological data cleaning for detecting outliers and duplicates [6], and to Gene Ontology for finding relationships among terms of the three ontology levels (cellular components, molecular functions and biological processes) [9], [10], but not for finding constraints or dependencies or anomalies.

With the use of association rules we find causality relationships among attribute values. Then, by analyzing the support and confidence of each rule, (probabilistic) data constraints and

functional dependencies may be detected. They may both show the presence of erroneous data or highlight novel semantic information.

2 Background

There are several ways to describe a database at a logic level (hierarchical, relational, reticular or object oriented). Now the most common type is the relational database, in which every item of information is represented by a *relation*, i.e. a table. In the biological domain there are examples of relational databases, such as cuticleDB (http://biophysics.biol.uoa.gr/cuticle), RCSB Protein Data Bank (http://pdbbeta.rcsb.org), Identitag (http://pbil.univ-lyon1.fr/software/identitag), AbMiner (http://discover.nci.nih.gov/abminer), PfamRDB (http://sanger.ac.uk/pub/databases/Pfam), and Reactome (http://www.reactome.org).

For databases created some years ago, without a relational structure, there are now a lot of tools to parse and load their data into a relational schema, since it is more convenient to work with tables rather than others data representations. For example the BioWarehouse toolkit (http://brg.ai.sri.com/biowarehouse) enables to translate the flat file representation of some databases, such as SwissProt (www.ebi.ac.uk/swissprot), NCBI (www.ncbi.nih.gov), KEGG (www.genome.jp/kegg) and GO (www.geneontology.org), into a relational schema.

A *relation* is composed by many *tuples* (rows), each of which represents an instance of an *entity*, and many columns, which represent the *attributes* of the entity. For example, in a protein database, the protein is an entity and its structures, function and sequence are its attributes. Every row related to a specific protein with all its attribute values is an instance of the protein entity. The table that contains all the proteins with their attributes is a relation.

The relational model is characterized by a structured, fixed format, since data values have to be homogeneous and have to meet several constraints. The database structure can be known or unknown in advance; in both cases our analysis is helpful, in order to determine the unknown structure or to investigate data constraints, which allows to discover novel information and errors.

Data constraints could be divided in two groups: domain constraints and tuple constraints. The first ones limit possible values of single attributes, the second ones consider relationships among values of different attributes of the same tuple. A typical example of domain constraint is *(age≥0)*. Instead an example of tuple constraint could be *(NOT (merit=yes) OR (mark=30))*: it means that a person can't gain the merit if his/her mark is not 30.

Data dependencies add semantics to a database schema and are useful for studying various problems such as database design, query optimization and dimensional reduction. A functional dependency states that if in a relation two rows agree on the value of a set of attributes X then they must agree on the value of a set of attributes Y. The dependency is written as $X \rightarrow Y$. For example, in a relation such as *Buyers (Name, Address, City, Nation, Age, Product)*, there is a functional dependency *City* → *Nation*, because for each row the value of the attribute *City* identifies the value of attribute *Nation*.

In this paper we consider tuple constraints and functional dependencies, since they lead to specific semantic errors such as discrepancies and inconsistencies, and in particular functional constraint violations.

3 Discovering constraints

Most existing data cleaning works focus on the problem of removing duplicates or dealing with syntactic errors [3], [4], [5]. Our method, instead, is useful to understand the database

structure and the relationships among attributes. It improves the knowledge of the domain in which it is used. Moreover it is useful to discover semantic anomalies, especially inconsistencies due to data constraint and functional dependency violations.

Figure 1 synthesizes the phases of the proposed framework. We first apply our method to discover tuple constraints and functional dependencies by means of association rules found by the *Apriori* algorithm [7]. A detailed description about our method and its goals will be presented further in this section. Since functional dependencies and tuple constraints have been found, we test our results in order to verify the method accuracy. Finally, we analyze the obtained results and show how they can be used to improve domain knowledge and data quality.

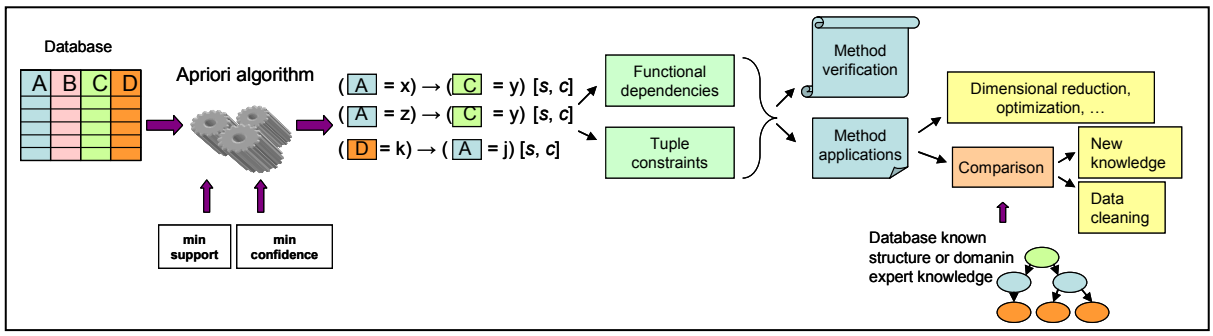

Figure 1 – Schema of the proposed framework

If constraints and dependencies of a database are known, they can be used as criteria to evaluate its accuracy. If a tuple does not satisfy a tuple constraint or a functional dependency, its data is not correct. In biological databases such constraints and dependencies are often unknown or incomplete, due to the complexity of stored information. We propose a method to discover them by means of association rule mining.

Association rule discovery is an important data mining technique, which is commonly used for local pattern detection in unsupervised learning systems. They show attribute value conditions that occur together in a given dataset, which are useful for finding correlations among sets of items in transactional or relational databases.

The rules are expressed in the form: *body* → *head [support, confidence]*. *Body* and *head* are conjunctions of pairs (attribute=value). *Support* (*s*) and *confidence* (*c*) are computed as:

(1) $$s = \frac{n_{bh}}{n}$$

(2) $$c = \frac{n_{bh}}{n_b}$$

where n_{bh} is the number of data instances that contain both body and head, n is the total number of data, and n_b is the total number of data instances containing the body [1]. n_{bh} is also called absolute support, while the support defined by formula (2) is a relative value with respect to the total number of tuples.

For example, the rule *((City=Paris)^(Age=30)) → (Product=Car) [0.5%, 60%]* means that thirty years old people living in Paris that bought a car are the 0.5% of the buyers stored in the database. It means also that 60% of thirty years old people living in Paris bought a car.

The concepts of support and confidence are used to determine the frequency and the importance of the discovered rules. Association rules that have confidence equal to 1 are tuple constraints, since they represent value constraints for those tuple attributes examined in the head and in the body of the rule.

In the special case in which all the association rules that involve the same attributes have confidence equal to 1 and the sum of all association rule supports is 1, a functional dependency between the corresponding elements is found [8]. The following formula is verified:

$$\sum_{i \in A} s_i = 1 \tag{3}$$

where A is the set of all association rules representing tuple constraints on the same attributes, and s_i is the support of every such rule.

In the example of *buyers* database, we may find the following rules

(City=Paris)→(Nation=France) [20%, 100%], *(City=Turin)→(Nation=Italy) [50%, 100%]*, *(City=London)→(Nation=United Kingdom) [30%, 100%]*.

In this case, the sum of the supports of all the rules that have *City* in the body and *Nation* in the head is

$$\sum_{City \to Nation} s_i = 0.2 + 0.5 + 0.3 = 1 \tag{4}$$

For these reasons we can also deduce that *City* → *Nation* is a functional dependency.

The dependencies between attributes can be expressed by considering confidence and support in a single expression [8], as in the following formula

$$p = \sum_{i \in A} s_i \cdot c_i \tag{5}$$

where p is an index of the dependency degree. When p is equal to 1, there is a functional dependency between attributes. Note that if the confidence value is not equal to 1, the p index will never be equal to 1.

Sometimes rules with confidence close to 1 or with the sum of supports near to 1 are found. For example, given the following rules

(City=Paris)→(Nation=France) [19%, 95%], *(City=Paris)→(Nation=Australia) [1%, 5%]*,

the issue is: "Are the records with people living in Paris, Australia wrong, or does a city named Paris exist in Australia?", or also: "Is *(NOT (City=Paris) OR (Nation=France))* a tuple constraint and *City* → *Nation* a functional dependency?". In these cases a domain expert opinion is needed.

In this work we aim at discovering every tuple constraint and functional dependency in a given database. We further consider every association rule with confidence higher than a minimum confidence threshold, and with the sum of supports higher than a minimum support threshold, in order to investigate errors or anomalies in the system as explained in Section 4.1.

3.1 Extracting association rules

The first step in finding association rules is to look for attribute values that appear in the same tuple. Every couple of attribute-value is an *item*. A set of items is called *itemset*. We are interested in finding *frequent* itemset, which are itemset with support higher than a specific threshold. A commonly used algorithm for doing this is the Apriori algorithm [7]. The algorithm relies upon a fundamental property of frequent itemsets, called the *apriori property*: every subset of a frequent itemset must also be a frequent itemset. The algorithm proceeds iteratively, first identifying frequent itemsets containing a single item. In subsequent iterations, frequent itemsets with n items identified in the previous iteration are combined

together to obtain itemset with *n+1* items. A single scan of the database after each iteration suffices to determine which generated candidates are frequent itemsets.

Once the largest frequent itemsets are identified, each of them can be subdivided into smaller itemsets to find association rules. For every largest frequent itemset *s*, all non-empty subsets *a* are computed. For every such subset, a rule $a \rightarrow (s-a)$ is generated and its confidence is computed. If the rule confidence exceeds a specified minimum threshold, the rule is included in the result set.

4 Experimental results

In biological databases functional dependencies are not known, or sometimes are incomplete, due to the complexity of stored data. It is possible that none of the discovered rules have a confidence equal to 1, due to the errors that could be present in the database. Our aim is to identify rules with confidence close to 1, according with a given tolerance, since they could represent errors to clean or anomalies in the system to analyse. The tolerance parameter is discussed in Section 4.1.

We chose two databases whose structure and dependencies among data were known in order to verify the accuracy of our method. We considered SCOP (Structural Classification Of Proteins, http://scop.berkeley.edu) and CATH (Class, Architecture, Topology and Homologous superfamily, version 3.0.0, http://www.cathdb.info) databases, which are characterized by a hierarchical structure, similarly to many biological data sources. However, our method is independent of the hierarchical structure, since the Apriori algorithm and our analysis can be applied to different data models ranging from the relational model to XML.

The SCOP database classifies about 30.000 proteins in a hierarchical tree of seven levels. From the root to the leaves they are: class, fold, superfamily, family, protein, and species, as shown in Figure 2. This tree structure is particularly suitable for verifying the tuple constraints and functional dependencies that can be extracted by our tool. In fact, in this database tuple constraints are represented by tree edges, while functional dependencies are represented by the tree hierarchy.

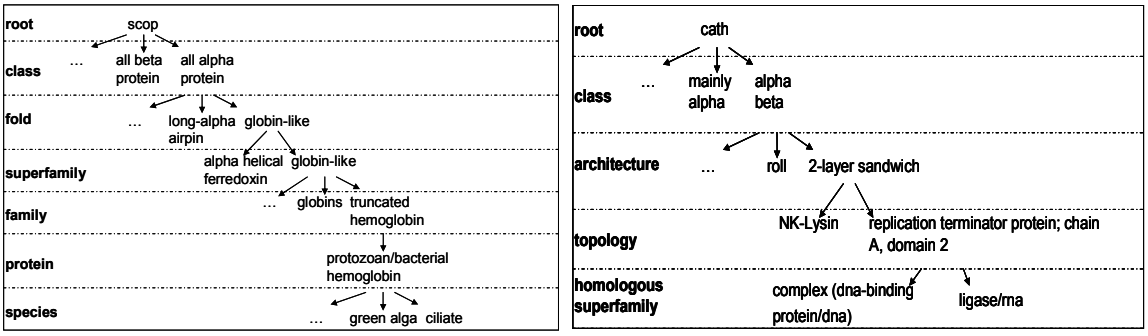

Figure 2 – SCOP (on the left) and CATH (on the right) hierarchical trees

For example we expect to find the following association rules:

(Superfamily=alpha helical ferredoxin)→(Fold=globin-like) with confidence of 100%

(Superfamily=globin-like)→(Fold=globin-like) with confidence of 100%

and $\sum_{Superfamily \rightarrow Fold} s_i = 1$

In this way we can deduce that *(NOT(Superfamily=alpha helical ferredoxin) OR (Fold=globin-like))* and *(NOT(Superfamily=globin-like) OR (Fold=globin-like))* are tuple constraints and *Superfamily→Fold* is a functional dependency.

The CATH database is a hierarchical classification of protein domain structures in the Protein Data Bank (http://www.rcsb.org/pdb/). Protein structures are classified using a combination of automated and manual procedures. There are four major levels in this hierarchy: Class, Architecture, Topology and Homologous superfamily, as shown in Figure 3. Domains within each Homologous superfamily level are subclustered into sequence families using multi-linkage clustering, identifying five family levels (named S, O, L, I, D). Thus, the complete classification hierarchy consists of nine levels (CATHSOLID).

The actual version of the CATH database (release 3.0.0, May 2006) includes 4 Classes, 40 Architectures, 1110 Topologies, 2147 H-superfamilies, and 86151 total protein domains.

4.1 Experimental setup

To set suitable values for the support and confidence parameters, we examined their meaning in different cases.

For the purpose of identifying functional dependencies both the confidence and the sum of supports must be equal to 1 (see formula (4)). For tuple constraints, the minimum confidence must be equal to 1. The minimum support value allows us to concentrate on the most frequent constraints. If the support is set to the inverse of the total number of records, then all the constraints are considered (this support corresponds to rules contained in a single data entry).

Since interesting anomalies generally affect a relatively small subset of records in a database, the minimum support and confidence must be low enough to detect all these data. Unfortunately, mining the association rules with low confidence is unfeasible on large databases because it is computationally too intensive. Furthermore, setting the confidence to low values is useless because the identified items include both actual anomalies and many other irrelevant features, which do not represent dependency exceptions. To solve this problem, we find rules characterized by a high confidence in order to investigate records that do not respect such rules. We call such records anomalies.

In order to detect anomalies, we performed several experiments by setting the support to the lowest value (the inverse of the total number of records) and by varying the confidence in the range from 0.999 to 0.900. We defined tolerance as the complement of the confidence value (e.g., confidence = 0.950, tolerance = 0.050). Results on anomaly detection will be provided in Section 4.3.

attributes		original	faulted
cf	***cl***	1.000000	**0.999774**
dm	***cf***	1.000000	**0.999972**
dm	*cl*	1.000000	1.000000
dm	***fa***	1.000000	**0.999944**
dm	***sf***	1.000000	**0.999945**
fa	***cf***	1.000000	**0.999972**
fa	*cl*	1.000000	1.000000
fa	***sf***	1.000000	**0.999944**
px	*cf*	1.000000	1.000000
px	*cl*	1.000000	1.000000
px	*dm*	1.000000	1.000000
px	*fa*	1.000000	1.000000
px	*sf*	1.000000	1.000000
px	*sp*	1.000000	1.000000
sf	***cf***	1.000000	**0.999916**
sf	*cl*	1.000000	1.000000
sp	***cf***	1.000000	**0.999972**
sp	*cl*	1.000000	1.000000
sp	***dm***	1.000000	**0.999817**
sp	***fa***	1.000000	**0.999972**
sp	***sf***	1.000000	**0.999945**

Table 1 - functional dependencies among attributes in the SCOP database

4.2 Functional dependency identification

We validated our approach by means of two steps: (i) verification that the proposed method correctly identifies all and only the functional dependencies and the tuple constraints contained in the database; (ii) verification that our algorithm actually detects violations of such constraints. In both cases we exploited the structural knowledge of the examined databases only for evaluating the obtained results.

To perform the first step we executed the algorithm with the lowest support value and setting the confidence to 1. To show that the result is sound and complete, we separately computed all the known tuple constraints and functional dependencies

from the database structure, as explained at the beginning of Section 4. Our method detected all and only the same 519781 tuple constraints contained in the SCOP database and the 239483 ones contained in the CATH database. Furthermore, by applying formula (3) on the detected tuple constraints, we identified all the functional dependencies contained in the databases.

In the second step we verified that the proposed algorithm actually detects erroneous data. For this purpose, we performed a simulation of fault injection tests. We changed random values at different levels of the hierarchy, by substituting the actual value with one randomly taken from another branch of the tree. This kind of misclassification is rather subtle to identify, since the value is acceptable (i.e., it is valid and it is not misspelled), but it assigns a particular protein to the wrong class for one or more levels of the hierarchy.

Table 1 shows the results of one of the fault injection tests on the SCOP database. The first two columns contain the attributes which are functionally dependent; their names are explained in the example below. The third column represents the results obtained applying formula (4) on the original data. Finally, the fourth column represents the same kind of results on SCOP data after a fault injection simulation where one random fault has been injected for some levels of the classification hierarchy. All the affected attributes represented in bold in the first two columns report dependencies whose value falls below 1 and our method allows us to detect all of the faults. The same results have been obtained analyzing the CATH database.

Example

In the SCOP database, the record of the protein chain *px*=100068 is composed by the following attributes: class *cl*=46456 (alpha proteins), fold *cf*=46457 (globin-like), super family *sf*=46458 (globin-like again), family *fa*=46459 (truncated hemoglobin), protein domain *dm*=46460 (hemoglobin) and species *sp*=46461 (ciliate). One of the injected faults is the misclassification of the record by assigning the value *fa*=46463 (globins) instead of *fa*=46459 (truncated hemoglobin) to the family attribute. The faulty value *fa*=46463 (globins) actually exists in the database and is the correct family attribute of many other proteins.

To identify the misclassified records, we analyze the rules with confidence below 1. In this example, the rule protein domain *dm*=46640 → family *fa*=46459 occurs in 26 records with confidence=0.963. We classified records that do not respect such rule as anomalies. Thus, the records with protein domain *dm*=46640 and family *fa*≠46459 are selected as either candidate inconsistency or information for further investigation by biological experts.

4.3 Anomaly detection

In the previous section we showed that the proposed method is able to recognize erroneous data and all the functional dependencies and tuple constraints in the database. In this section we concentrate our analysis on the detection of the anomalies that could be contained in the database. These anomalies could be inconsistencies in the data or biological correct, albeit infrequent, situations. We called these infrequent events *biological exceptions*, interesting for further analyses.

We performed experiments on the original data with tolerance values ranging from 0.001 to 0.1 (corresponding to confidence values from 0.999 to 0.90). The confidence value must be lower than 1 but high enough to assure that mainly such anomalies are detected, while the support value has been set to the inverse of the total number of records, as discussed in Section 4.1. Results show a consistent number of anomalies whose presence increases as the tolerance value of the corresponding tuple constraint increases (see Figure 3).

The tolerance parameter allows us to investigate the meaning of the detected anomalies. For instance, if the tolerance is low (e.g. 0.01), each detected anomaly affects only few records which represent strong exceptions in the data.

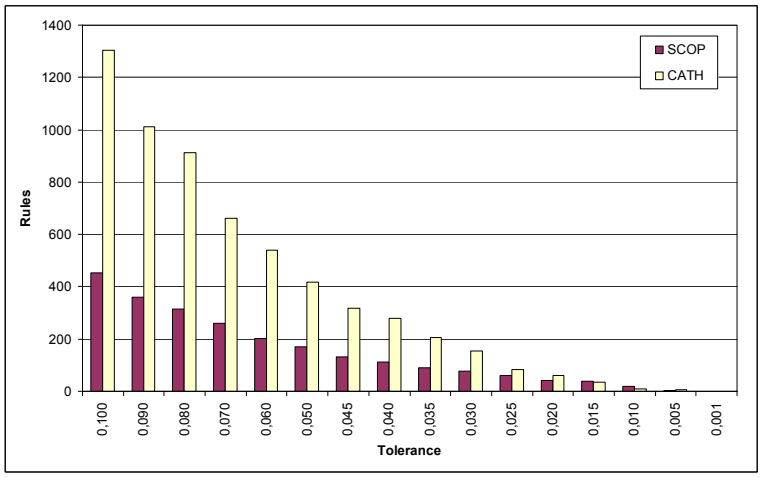

Figure 3 - Number of anomaly-rules found for different values of tolerance

Anomalies detected by our algorithm allow domain experts interested in the domain to focus their analysis on a small set of data in order to highlight biological exceptions or inconsistencies in the data. To distinguish between them, we further analyzed the discovered anomalies by means of three approaches: querying different related databases (e.g., GO, PDB, Swiss Prot, CATH, SCOP), searching relevant information in literature, comparing the obtained results with the examined database structures. In this work we perform this investigation manually, but as a future work we plan to extend our method with a framework which automatically performs a distributed retrieval of related information from different databases and a comparison of same information with the examined database structure. In fact, if the database structure is known, errors can be distinguished from exceptions automatically, by comparing the results with the structure. If the attribute values of a tuple do not respect the attribute structure, we can conclude that it is an error. Otherwise it is a biological exception.

The following example shows an anomaly analysis that aims to distinguish between biological exceptions and errors. First, a database research is done, followed by a literature research. All of these steps are here executed manually. We detected some infrequent event (namely, *anomaly*) during the experiments on SCOP and, in order to discuss our anomaly investigation approach, we focused our analysis on a few of them. Searching for the same anomalies on other databases (i.e., Swiss-Prot, PDB, GO, CATH), we found out that such anomalies are biological correct, albeit infrequent, situations. In addition, by applying our full algorithm to the CATH database we discovered the same data as infrequent events. This is a further proof of the consistency of the proposed method.

For space reason, in the example below we report only a single case of anomaly analysis, but we performed similar investigation on other anomalies in SCOP and in CATH reaching similar results.

Finally, since in the SCOP and CATH case the database structure is known, we compared our results with those structures in order to obtain a further validation of our approach.

Example

In the SCOP database, one of the anomalies reported by our method indicates that the Ntn hydrolase-like fold has the hypothetical protein MTH1020 superfamily only in one tuple

(tolerance = 0.00229), while it has the N-terminal nucleophile aminohydrolases superfamily in all the other 435 tuples (confidence = 0.99771).

Given such anomaly, we performed a query in other databases (i.e., GO, Swiss Prot, PDB, CATH) in order to investigate if it was related to an error or a biological exception. As result, we found out evidences that the SCOP classification for this protein was biologically correct, thus it was not an inconsistency. In particular, we verified in CATH that this protein is classified in the same *alpha+beta* class as in SCOP and that it has a 4-layer sandwitch architecture (and Glutamine topology). This architecture consists on the 4 layers *alpha/beta/beta/alpha* that are the same for the N-terminal nucleophile aminohydrolases fold found in the SCOP classification. Moreover, we found evidence in literature that the crystal structure of MTH1020 protein reveals an Ntn-hydrolase fold [11].

We applied our algorithm also to the CATH database and we discovered some anomalies (as reported in Figure 3), by setting the tolerance parameter within 0.005. Among these anomalies, we noticed the one for the hypothetical protein MTH1020. This result confirms the consistency of the proposed method for anomaly discovery.

Furthermore, since the SCOP structure is known, we compared the result with the database structure, confirming the correctness of this relationship (i.e. it is not an error).

5 Conclusions

In this paper we presented a framework for the application of data mining tools to data cleaning in the biological domain. We focused on tuple constraints and functional dependencies detection in representative biological databases by means of association rule mining. By analyzing association rules we can deduce not only constraints and dependencies, which provide structural knowledge on a dataset and may be useful to perform query optimization or dimensional reduction, but also the anomalies in the system, which could be errors or interesting information to highlight to domain experts.

We have applied our analysis technique to SCOP and CATH databases. We plan to extend our approach to different database models, such as XML or a collection of relational tables, and to integrate automatic distributed inquiry about the detected anomalies on such databases, in order to help domain experts to distinguish biological anomalies from errors. Further developments of this work include the application of our method to heterogeneous data sources, to derive schema information that may be exploited during data integration.

6 Acknowledgements

We would like to thank Paolo Garza for his help in association rule extraction and for many stimulating discussions.

7 References

[1] J. Han, M. Kamber. Data Mining: Concepts and Techniques. 2006.

[2] H. Müller, F. Naumann and J-C. Freytag. Data quality in genome databases. Proceedings of the International Conference on Information Quality (IQ 2003), Boston, 2003.

[3] H. Galhardas et al. AJAX: An Extensible Data Cleaning Tool. Proc. 2000 ACM SIGMOD Conf. Management of Data (SIGMOD 00). ACM Press, 2000, p. 590.

[4] H. H. Shahri and A. A. Barforush. A Flexible Fuzzy Expert System for Fuzzy Duplicate Elimination in Data Cleaning. DEXA 2004, LNCS 3180, pp. 161 - 170.

[5] Mong Li Lee, Tok Wang Ling and Wai Lup Low. IntelliClean: A Knowledge-Based Intelligent Data Cleaner. KDD 2000, Boston.

[6] J.L.Y. Koh, et al. Duplicate Detection in Biological Data using Association Rule Mining. 2nd European Workshop on Data Mining and Text Mining for Bioinformatics. An ECML/PKDD 2004 workshop, Pisa, Italy, September 24, 2004.

[7] R. Agrawal, R. Srikant. Fast Algorithms for Mining Association Rules. VLDB Conference, 1994, Santiago, Cile.

[8] E. Baralis, P. Garza, E. Quintarelli, L. Tanca. Answering Queries on XML Data by means of Association Rules. In Current Trends in Database Technology, vol. 3268, 2004.

[9] A. Kumar, B. Smith, C. Borgelt. Dependence Relationships between Gene Ontology Terms based on TIGR Gene Product Annotations. 3rd International Workshop on Computational Terminology (CompuTerm), 2004.

[10] O. Bodenreider, M. Aubry, A. Burgun. Non-lexical approaches to identifying Associative relations in the gene ontology. Pacific Symposium on Biocomputing 2005.

[11] V. Saridakis, D. Christendat, A. Thygesen, C.H. Arrowsmith, A.M. Edwards, E.F. Pai. Crystal structure of Methanobacterium thermoautotrophicum conserved protein MTH1020 reveals an NTN-hydrolase fold. Proteins 2002 Jul 1;48(1):141-143