A Machine Learning Approach to Foreign Key Discovery

Alexandra Rostin[1] Oliver Albrecht[1] Jana Bauckmann[2]
Felix Naumann[2] Ulf Leser[1]
[1]Humboldt-Universität zu Berlin, Berlin, Germany, {rostin,oalbrecht,leser}@informatik.hu-berlin.de
[2]Hasso-Plattner-Institut, Potsdam, Germany {Jana.Bauckmann,Felix.Naumann}@hpi.uni-potsdam.de

ABSTRACT
We study the problem of automatically discovering semantic associations between schema elements, namely foreign keys. This problem is important in all applications where data sets need to be integrated that are structured in tables but without explicit foreign key constraints. If such constraints could be recovered automatically, querying and integrating such databases would become much easier. Clearly, one may find candidates for foreign key constraints in a given database instance by computing all inclusion dependencies (IND) between attributes. However, this set usually contains many false positives due to spurious set inclusions. We present a machine learning approach to tackle this problem. We first compute all INDs of a given schema and let each be judged by a binary classification algorithm using a small set of features that can be derived efficiently using standard SQL. We demonstrate the feasibility of this approach using cross-validation with several state-of-the-art classification algorithms. With the J48 algorithm, our approach consistently reaches F-measures above 80% and often close to 100% as evaluated on six different data sets from three different domains.

1. Introduction
A basic requirement for integrating databases into a unified system is to know and to understand the database's structure and semantics. For instance, when building a mediator based system using methods as, for instance, described in [15, 19], domain experts first must identify semantic correspondences between the local and global schemas [22]. Obviously, they must be aware of the structure and – at a higher level – the semantics of the databases to correctly solve this task.

Understanding source schemas often is a time-consuming and costly process [10], which led to a growing interest in automatic methods for supporting database integration [9]. Accordingly, much research has been devoted to automatically find correspondences between schema elements in different database schemas (schema matching; see Section 5). Surprisingly, much less attention has been devoted to the problem of identifying relationships between elements within a schema, although this knowledge is equally important for building high-quality integrated systems.

One important class of relationships between elements of a schema are foreign keys or, more precisely, foreign key constraints (FKC). A FKC essentially states that tuples of a relation (containing the foreign key) are dependent on tuples of another relation (containing the primary key); it thus represents a semantic relationship between tuples. However, in many areas databases do not provide foreign key definitions [13]. This absence may have several reasons, such as the lack of support for checking foreign key constraints in the host system (e.g., MySQL supports foreign key constraints only in one of its container types and even there only under certain rather awkward conditions), the fear that checking such constraints would impede database performance, or a simple lack of database knowledge within the development team. All these reasons are, for instance, quite frequently met in Life Science databases. The very popular genome database Ensembl [8] is shipped as MySQL dump files with more than 200 tables but not a single foreign key constraint (probably due to the above mentioned problems with foreign key constraints in MySQL); equally, the MSD database of protein structures [21] is available as dump files for a commercial RDBMS and contains more than 150 tables but no foreign key definitions. Understanding and integrating such databases would greatly benefit from a method to recover foreign keys from the data. Besides its value for integration, knowing about such intra-schema relationships is also very important for database exploration [5].

In this paper, we study the problem of automatically detecting putative foreign key constraints in database instances that lack of primary key (PK) and foreign key (FK) constraints. A FKC between a foreign key A and a primary key B implies that all values contained in A are also present in B. A pair of attributes where the values of the one attribute are contained in the set of values of the other attribute underlie an *inclusion dependency* (IND). However, set inclusion only is a necessary but not a sufficient characteristic for a FKC, because the inclusion of value sets may occur by pure chance. For instance, in the MSD database there are >40,000 INDs, of which >5,000 have a unique target attribute and are therefore potential FKs. Many of the non-foreign-key INDs stem from tables storing a controlled vocabulary that is referenced by a surrogate key, often numbers starting by 1 in each table. Obviously, the keys of different such CV tables often value-include each other without any existing semantic relationship. Or consider a Boolean attribute containing only 0s and 1s. Such an attribute almost certainly induces an IND with most Integer attributes.

Unfortunately, foreign keys are semantic relationships and cannot be inferred with certainty from an instance of a schema alone. Fortunately, a FKC often exhibits certain characteristics that distinguish it from a spurious IND. For instance, the set of values of a foreign key often covers almost all values of its primary key, and a FK attribute name often contains (exactly or approximately) the name of its PK, possibly preceded or followed with certain substring such as "_ID". Naturally, such evidences can only be considered as soft filters and do not prove, but increase or decrease the trust in a given IND to be a FKC.

12th International Workshop on the Web and Databases (WebDB 2009)

Building on this observation, we developed and tested a machine learning approach to detect probable foreign keys. We defined a set of characteristics and used them as features, which are fed into a state-of-the-art classification algorithm to classify an IND as FKC or not. We evaluated this approach using cross-validation on six different databases from different domains using four different classification methods. Overall, the results show that foreign key discovery can be tackled quite successfully with machine learning techniques and that our set of features is well capable of separating foreign key constraints from other INDs.

The rest of this paper is structured as follows: In the next section, we give an overview of our system. Section 3 describes our set of features to classify INDs as FKC or not. Section 4 shows the results of evaluating our approach on different data sets. We discuss related work in Section 5 and conclude in Section 6.

2. System Overview

We built a system that aims to find in a relational database all pairs of attributes where one attribute is a primary key and the other attribute is a corresponding foreign key. To this end, we regard only the values and names of schema elements. With a slight abuse of notation, we call such pairs *foreign key constraints (FKC)*, while we call all INDs that are not FKC *spurious INDs*. Our system first learns a "model" to tell FKCs from spurious INDs using a set of computed properties of positive and negative examples. This model is later used to classify each new pair of attributes.

In the learning phase, we analyze a set of databases where all FKC are known. We determine all INDs (see below) and compute a feature vector for each (see next section). Pairs that are FKC are labeled positively, other pairs are labeled negatively. We present this set of labeled feature vectors to a machine learning algorithm, which turns it into a model.

In the application phase, we are given a database without FKC and want to recover them. Again, we first filter the set of all pairs of attributes to those that form an IND, since only those are candidates for PK-FK relationships. We compute the feature vector for each IND and present it to a classifier, which in turn judges the feature vector using its previously learned model. For evaluation, we apply this method to a database for which we know all FKC and count all false positives, false negatives, true positives, and true negatives.

We determine all INDs using the SPIDER algorithm [1]. Because detecting all INDs in principle requires a test of all pairs of attributes, it is vital to have an efficient algorithm at hand to be able to handle databases of non-trivial size. SPIDER uses three main tricks to speed up the comparison: First, it persists all attribute value lists in a sorted manner; second, it tests all pairs concurrently with a single sweep over all values lists; third, it exploits the fact that a single value in the dependent attribute that is not contained in the referenced attribute is sufficient to exclude this pair. Using SPIDER we can, for instance, analyze the entire MSD database (32GB, 2.713 columns, 176 tables) in ~4 hours to find all of its 40,415 inclusion dependencies. As already stated, most of these INDs are spurious.

We want to stress again that the detection of INDs is a problem that can be solved exactly, while the detection of FKC can only be solved heuristically. An IND is a necessary precondition for a FKC, but only a human expert can "promote" an IND to a FKC, because INDs can appear by coincidence. In contrast, FKC are consciously specified and denote a semantic relationship.

3. Features for Classifying INDs

In this work, features are properties of an IND *(A, B)* whose values are used to give hints on whether the IND represents a FKC or not. Here, *B* is the primary key (referenced attribute) and *A* is the foreign key (dependent attribute). Since it is generally acknowledged that in many applications, the choice of features has more influence on the achievable performance than the choice of classification method [11], we performed an extensive manual study to find meaningful features by using common sense and by carefully studying positive and negative examples.

We derived a set of ten different features. These features are listed here, each followed by a brief explanation of its underlying rationale. Let *name*(A), *name*(B) denote the attribute names of *A* and *B*, and let *s*(A), *s*(B) denote the set of distinct values of A and B, respectively.

- **DistinctDependentValues** (F1): The cardinality of *s*(A). Usually, attributes that are foreign keys contain at least some different values, as otherwise almost none of the referenced objects have a dependent value.
- **Coverage** (F2): The ratio of values in *s*(B) that are contained in *s*(A) to all values in *s*(A). Usually, foreign keys cover a considerable number of primary keys.
- **DependentAndReferenced** (F3): Counts how often the dependent attribute A appears as referenced attribute in the set of all INDs. Usually, an attribute that is a foreign key is not at the same time a primary key that is referenced as foreign key by other tables.
- **MultiDependent** (F4): Counts how often A appears as dependent attribute in the set of all INDs. If *s*(A) is contained in the set of values of many other attributes, the likelihood for each of these INDs being a FKC is decreased.
- **MultiReferenced** (F5): Counts how often B appears as referenced attribute in the set of all INDs. We assume that primary keys are often referenced by more than one foreign key; therefore, a high number for F5 raises the chances of the pair under consideration to represent a FKC.
- **ColumnName** (F6): Measures the similarity between *name*(A) and *name*(B), also considering the name of the table of which B is an attribute. We currently only check for exact matches or complete containment. Using more advanced string similarity measures would be an obvious improvement.
- **ValueLengthDiff** (F7): The difference between the average value length (as string) in *s*(A) and *s*(B). We expect that the average length of the values should be very similar whenever foreign keys reference a non-bias sample of the primary keys. They should be highly similar if every value of primary key is referenced (it will not be identical in 1:N relationships).
- **OutOfRange** (F8): The percentage of values in *s*(B) that are not within [min(*s*(A)), max(*s*(A))]. Usually, the dependent values should be more or the less evenly distributed over the referenced values and not only cover a small, continuous range.

- **TypicalNameSuffix** (F9): Checks whether *name*(A) ends with a substring that is an indication for foreign keys. We currently use only „id“, „key“, and „nr“ (German for “no” - number; one of our test schemas has German attribute names).
- **TableSizeRatio** (F10): The ratio of the number of tuples in A and the number of tuples in B. Usually dependent attributes do not reference only a very small subset of their primary keys.

We developed these features during an extensive phase of trial-and-error. For instance, we found that roughly 60% of all foreign keys in our evaluation data set cover all values of their referenced primary keys, and none covers less than 10%. Similarly, in almost all FKCs less than 5% of the values of B lie outside the min-max range spread by A (F8). Accordingly, both features seem to be good candidate for our purpose. We also evaluated all features using different feature selection methods, showing that all of them are useful under some settings (see Section 4.3).

Note that the features in general are not independent of each other. For instance, a low value in F1 usually implies a low value in F10 (given that B is not very small). Dealing with those dependencies between features is left to the classifier.

Computing feature vectors

Computing the feature values for all INDs of a given database is very fast. Features F3, F4, F5, F6, and F9 only analyze metadata and are thus independent of the number of values in the database. All other features are easily implemented as SQL queries using only standard SQL functionality. For instance, computing the F1 values for an IND requires nothing more than a count on all distinct values of the dependent attribute; furthermore, it needs to be computed only once for all attributes (and not once for each IND containing it). Since all necessary SQL queries are counts of single tables without joins, their execution is rather fast even when very large tables are involved. Note that computing the feature values could be made even faster when it would be embedded in the SPIDER algorithm. Also note that approximate values as kept by most systems within their database-internal statistics very likely would serve our purpose equally well (though we have not yet tested it).

4. Evaluation

We evaluated the ability of the chosen features to correctly classify INDs with four different classification algorithms. These were (1) Naïve Bayes (NB), (2) Support Vector Machines (SVM), (3) J48, and (4) Decision Tables (DT). We used the implementation of these algorithms as provided by the WEKA machine learning tool[1]. Details on the specific variants of these methods as they are implemented in WEKA can be found in the WEKA documentation.

4.1 Data Sets

We used six data sets from three different domains: three databases from the Life Sciences (SCOP, MSD, UniProt) [2], two databases storing information on movies (Movielens, Filmdienst), and the TPC-H benchmark. An overview of these databases and some of their properties is given in Table 1. All databases were downloaded as SQL dump or were transformed from flatfiles into a relational representation using publicly available parsers.

For datasets UniProt, Movielens, and TPC-H, foreign key constraints are provided with the download files. For Filmdienst, SCOP, and MSD we deduced them manually by analyzing the database documentation. However, in case of the MSD, we only checked 673 of the 5.431 INDs. Of those, 526 were manually classified as representing semantically meaningful foreign key relationships.

	Tabs	*Atts*	*Tuple*	*INDs*	*FKCs*	*Avg a/t*	*Min a/t*	*Max a/t*
UniProt	28	156	~8M	36	31	6	2	14
Filmdienst	14	92	>1M	79	15	8	2	32
Movielens	7	20	~1M	19	6	3	2	5
SCOP	4	22	~500K	11	5	6	2	12
TPC-H	8	61	~10M	33	9	8	3	16
MSD	176	2713	~300M	5431	(*)	15	3	94

Table 1. Properties of the six databases used for training / evaluation. ‘a/t’ means “attributes per table”.
(*) The total number of FKCs in the MSD is unknown

4.2 Feature Selection

We evaluated how well each of the features presented in Section 3 could help to distinguish FKC from spurious INDs by using feature selection. We performed this test using various methods implemented in WEKA. We omit detailed results here; see Table 2 for some examples. Features F2, F6, F7, and F8 were consistently under the top-selective features. However, all features were selected from at least one selection method. Therefore, we used the full feature set for the evaluation of the classifier performance described in the next section. We also performed tests with subsets of the features, but differences in performance were only marginal.

	F1	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**F10**
M1		X		X	X	X	X	X		X
M2	X	X		X		X		X	X	
M3	8	2	10	5	7	3	4	1	9	6
M4	8	3	10	6	7	2	4	1	9	5

Table 2. Results of different feature selection methods. ‘X’ means that method Mx has selected feature Fy as discriminative. M1: Best subset using ranked search; M2: same using randomized search; M3: Rank according to InfoGain; M4: Rank according to X^2-statistics.

4.3 Performance of Classifiers

We performed a number of tests to analyze the performance of the different classifiers. We give results of these tests in terms of F-Measure. F-Measure is the harmonic of precision and recall,

[1] http://www.cs.waikato.ac.nz/ml/weka/

[2] SCOP is a database of protein families; MSD is a reformatted version of the PDB protein database; UniProt is a protein sequence database, which we imported using BioSQL.

where precision is the percentage of correctly identified FKCs under all positively classified INDs, and recall is the number of recovered FKCs from the gold standard relative to all positively classified INDs. Because the differences between precision and recall were rather small (for instance, only 5% on average for J48), we omit them for the sake of brevity.

Machine learning approaches heavily depend on the data sets used for learning and their differences or similarities to the data sets used for evaluation. Showing the generality of results is an important issue. The method that is used most often is cross-validation, where a labeled data set is repeatedly split into a training set and an evaluation set. The performance of the classifier on unseen data is then estimated as the average performance of the different runs. There are various variations of this method regarding the proportion of data held back for testing versus training, the way how the single examples are sample etc. [11].

There are two ways how cross-validation could be applied to our problem: (1) Consider all INDs (recall that we classify INDs) as one data set, regardless of their original database. Applying this option to the D0 dataset (see Table 3) would mean that we randomly partition its 179 INDs into *k* sets and iteratively learn on *k*-1 partitions and evaluate on the *k*'th partition. (2) Take into account that different INDs stem from different databases and perform a cross-validation at database-level. Thus, INDs from one database are always used either only for training or only for evaluation.

ID	INDs	FKs	Dataset
D1	36	31	UniProt
D2	79	15	Filmdienst
D3	19	6	Movielens
D4	11	5	SCOP
D5	33	9	TPCH
DX	673	526	MSD

(a)

ID	INDs	FKs	Dataset
D6	142	35	D0 - Uniprot
D7	99	53	D0 - Filmdienst
D8	159	62	D0 - Movielens
D9	167	63	D0 - SCOP
DA	149	59	D0 - TPCH
D0	178	68	All - MSD

(b)

Table 3. Datasets used for training (b) and evaluation (a).

We decided to use a leave-one-out evaluation at the level of databases, i.e., option (2) (except for MSD, which we held back as an independent test dataset). This choice means that we systematically used all INDs from all but one database for learning and evaluated the learned model on the INDs of the missing database. We favored this approach over option (1), because we hypothesized that databases differ significantly in the way how the data underlying an FKC look like and thus in the properties that discern a spurious IND from a true foreign key relationship. Thus, if we used a training set that contained examples from all databases, the classifier would be able to learn the particularities of all databases and would thus have a big advantage during evaluation compared to the situation when it would be evaluated on an entirely new database. Moreover, the usual application scenario is seeing a new database, not seeing a new IND in an already known database. Therefore, cross-validation on the IND level would lead to overly optimistic estimates of the classifier performance in real-life scenarios.

This evaluation plan leads to the datasets shown in Table 3. Essentially, we built ten datasets, where five are made from four databases and five are made from just one. Furthermore, we built dataset D0, which contains all databases except MSD, and dataset DX, which consists only of MSD, for a final test.

Results of the resulting five experiments with using four classifiers are shown in Table 4. There are a number of interesting observations. Overall, the worst performance is 0.71, whereas the best performance is 1. The decision-tree-like algorithms J48 and DecisionTable have better results on average and also gather the best results most often, but for classifying the D1 set (UniProt), SVM achieves the best results. The smallest database, SCOP (D4), is classified perfectly by all methods; however, there is no general tendency that results would be better with more or less FKCs or with a larger or smaller ratio of FKCs to spurious INDs. The differences in the results show that there are clear differences in the way how different databases use foreign keys. For instance, Movielens is quite different from the other four databases, because the model learned only on those four reaches an F-measure of 0.805 on average[3].

DS for learning / evaluation	*Naive Bayes*	*SVM*	*J48*	*DecisionTab*	*Avg*
D6 / D1	0.86	**0.92**	0.84	0.8	0.855
D7 / D2	0.80	0.86	0.86	**0.93**	0.817
D8 / D3	0.71	0.71	**1.0**	0.8	0.805
D9 / D4	**1.0**	**1.0**	**1.0**	**1.0**	1.0
DA / D5	0.86	0.90	**0.95**	**0.95**	0.915
Average	0.846	0.78	**0.930**	0.896	

Table 4. Results (F-Measure) of four different classifiers on five different datasets. Best results per row are in bold.

We also performed an experiment where we learned from the combined dataset D0 and evaluated the results on the withheld dataset DX. This test is different from the previous ones as MSD was not used during the feature engineering phase; thus, while we tried to capture properties of FKCs in the five databases with our feature sets, this was not the case for the MSD database. Therefore, this experiment is more realistic than the previous ones. The results, shown in Table 5, are rather encouraging and show that our features seem to generalize quite well.

Finally, we wanted to explicitly test our hypothesis that databases have different characteristics. To this end, we evaluated on the DX dataset using J48 with two models:

- learning on the D0 dataset, and

[3] All databases are classified perfectly when the learning data includes the evaluation data, which shows that the instances are classifiable (data not shown).

- learning on a dataset that contains all INDs from D0 and a randomly chosen 10% sample of the 673 INDs from MSD (we call this dataset D0+).

In the first case, J48 reaches an F-measure of 0.78 (see Table 5). When learning on the D0+ dataset, J48 reaches an F-measure of 0.99. Thus, adding only a very small sample of all INDs of a database was, in this case, sufficient for the classifier to build a much better model.

Dataset for learning / evaluation	*Naive Bayes*	*J48*	*DecisionTab*
D0 / DX	0.84	0.78	0.79

Table 5. Results (F-Measure) from learning from UniProt, Filmdienst, SCOP, TPC-H and evaluating on the MSD.

For illustration, we show the complete decision tree that J48 generated from the D0+ dataset in Figure 1. In accordance to Table 2, features F6 (ColumnName) and F8 (OutOfRange) are most discriminative.

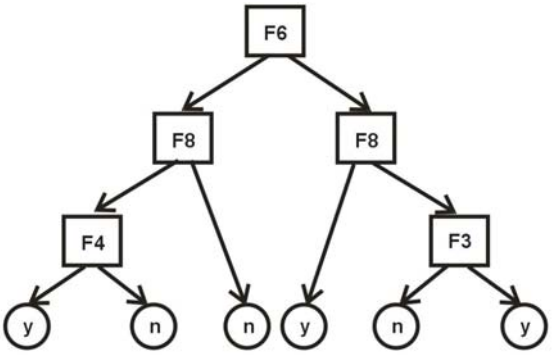

Figure 1. Decision tree computed by the J48 algorithm, learned on dataset D0+.

4.4 Error Analysis

We performed a preliminary analysis of the cases in which many classifiers make errors. We observed a number of different situations, some of which might be overcome by an improved set of heuristics, some not. The following is a list of situations where the classification-based approach errs for many classifiers:

- **Empty tables**. One of the schemas (UniProt) contains a FKC defined on an empty table. This attribute is never included in an IND and is thus never classified. Detecting such cases will be hard, but, on the other hand, it is doubtful whether it is sensible trying to do so.
- **Transitive foreign keys**. We observed the situation that an attribute B is a foreign key referencing an attribute A while B itself is referenced as primary key by a third attribute C. This rather unusual case is covered by F3. However, the problem here was not that the classifier erred in classifying (B,A), but it also classified (C,A) as FKC. Thus, our method detected a semantically meaningful yet technically redundant FKC that was not specified in the schema and thus counts as false positive during evaluation.
- **One-to-one relationships**. Whenever a 1:1 relationship between two tables (with attributes A, B) holds, our method will judge both (A, B) and (B, A) as IND and will very likely also classify both as FKC. Semantically, this makes sense, but bidirectional FKC are not supported in most RDBMS and thus, again, one of the directions will appear as false positive.
- **Small tables**. Some of the schemas contain tables that have only few tuples which proved difficult to classify. For instance, the Filmdienst database has a table for awards referencing the actors table; however, as only a very small fraction of all actors receive an award (that is represented in Filmdienst), the dependent attribute has only very low coverage of the referenced attribute and additionally a high OutOfRange percentage. Thus, it was misclassified frequently.

Many remaining errors were related to surrogate keys. For instance, the TPC-H schema has a column storing quantities of goods; the concrete values are randomly generated from the TPC data generator as small integers and grossly overlap with the surrogate keys of other tables with few tuples, such as suppliers.

5. Related Work

In [14] we described the Aladin project and its vision of a domain-specific data integration system that performs as much of the integration steps as possible using fully automatic techniques. It targets a specific type of integration in that it does not try to create a unified schema, but "only" tries to automatically link semantically associated schema elements, both within databases (intra-schema) and across databases (inter-schema). Such relationships support query rewriting, data browsing, or data analysis [3]. Clearly, we expect that reliance on automatic techniques inevitably introduces a certain level of error into the system. The overall research goal of our project, into which the present work fits, is precisely to study how low one can push this error on various tasks.

Foreign key constraints are a special case of a semantic relationship between attributes. Finding relationships between attributes has been studied intensively in the schema matching community. Approaches can be classified into schema-based (those that use the names and structure of schema elements), instance-based (those that use the data stored under schema elements), and hybrid approaches [20]. However, this line of research was and still is mostly devoted to the inter-schema case, while we target the intra-schema case and only a very specific question therein. Furthermore, schema matching approaches concentrate on semantically equivalent attributes, which is different from our case. Nevertheless, there are several projects that propose machine learning for schema matching [2, 6, 17], which also inspired our work. However, the types of features they use mostly are quite different (with name similarity being a notable exception), precisely because they compare attributes from different schemas, while we classify pairs of attributes from the same schema. Furthermore, we can exploit the fact that only INDs can classify as FKC, a restriction that is not applicable in the schema matching case.

The only other approach to the detection of foreign keys we are aware of is that of Lopes et al. [16]. It uses SQL workloads to deduce foreign keys under the assumption that join operations are usually (if not always) performed by equating a key with a foreign key. This approach completely depends on the availability of a sufficiently large workload. In data integration projects, such a workload is rarely available.

A related line of research to the problem we study is finding functional dependencies within a single relation, either for the exact or approximate case [12] or for the conditional case [7]. Similarly, the problem of computing INDs has been studied both exactly and approximately [1, 5, 18]. In these projects, the problem was attacked using an algorithmic approach that assumes a relationship to hold when all or a sufficiently large percentage of tuples of the database under study obey it. Brown and Haas study algebraic constraints between pairs of attributes to use them in query optimization [4]. In contrast to all these prior work, we follow a machine learning based approach to avoid fixed thresholds and to be able to exploit existing examples in a systematic manner.

6. Conclusions

Data integration is a perpetual problem on the web and is frequently associated with high cost, instable systems, and low data quality [10]. We take a step forward by reducing the high upfront cost: We presented a method that, given a database without any specified constraints, computes with high precision and high recall all PK–FK relationships. Clearly, this method could easily be extended to also detect relationships across databases, a step we are currently investigating.

We consider our study still in a preliminary state. One of the most important next steps is to test on more databases to be able to better judge the method's generality. If sufficient examples can be gathered, it would be interesting to see whether there are domain-dependent differences: Do, for instance, database developers deal differently with foreign keys than, say, developers of commercial ERP systems? Using more and more heterogeneous databases might also lead to the detection of new features. We also plan to study the trade-off between precision and recall, an aspect we mostly ignored in this paper. In its current setting, our method achieves roughly identical precision and recall, but in some applications it is important to tune a system either towards higher precision (i.e., when the results of the predictor are used in fully automatic systems sensitive to false positives) or higher recall (i.e., when the results are used in an interactive system where all proposed foreign key constraints are checked and approved by human experts). We are also working towards the detection of relationships between sets of attributes, i.e., non-unary inclusion dependencies and foreign key constraints.

Finally, we believe that a tool like the one we presented could also be very helpful in databases design. Actually, it can be used to suggest missing foreign keys in complex, populated schemas.

Acknowledgements

We thank Silke Trißl for manually evaluating more than 600 FK-candidates on the MSD schema, Tobias Flach for developing a GUI, and the WEKA team for providing their software.

References

[1] Bauckmann, J., et al. Efficiently Detecting Inclusion Dependencies. in International Conference on Data Engineering. 2007. Istanbul, Turkey.

[2] Berlin, J. and A. Motro. Database Schema Matching Using Machine Learning with Feature Selection.. in 14th CAiSE. 2002. Toronto, Canada.

[3] Bleiholder, J., et al., BioFast: Challenges in Exploring Linked Life Science Sources. SIGMOD Record, 2004. 33(2).

[4] Brown, P. and P.J. Haas. BHUNT: Automatic Discovery of Fuzzy Algebraic Constraints in Relational Data. in 29th International Conference on Very Large Databases 2003.

[5] Dasu, T., et al. Mining Database Structure: Or, How to Build a Data Quality Browser. SIGMOD. 2002. Madicon, USA.

[6] Doan, A., P. Domingos, and A. Halevy. Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach. in SIGMOD. 2001. Santa Barbara, CA.

[7] Fan, W., et al., Conditional functional dependencies for capturing data inconsistencies. ACM TODS, 2008. 33(2).

[8] Flicek, P., et al., Ensembl 2008. Nucleic Acids Res, 2008. 36(Database issue): p. D707-14.

[9] Halevy, A., M. Franklin, and D. Maier. Principles of Dataspace Systems. in PODS. 2006. Chicago, USA.

[10] Halevy, A., A. Rajaraman, and J. Ordille. Data Integration: The Teenage Years. in VLDB. 2006. Seoul, South Korea.

[11] Hastie, T., R. Tibshirani, and J. Friedman, The Elements of Statistical Learning. 2001: Springer.

[12] Huhtala, Y., et al., TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies. Computer Journal, 1999. 42(2): p. 100-111.

[13] Johnson, T., A. Marathe, and T. Dasu, Database Exploration and Bellman. IEEE Data Eng. Bull., 2003. 26(3): p. 34-39.

[14] Leser, U. and F. Naumann. (Almost) Hands-Off Information Integration for the Life Sciences. in Conference on Innovative Database Research (CIDR 2005). 2005. Asilomar, CA.

[15] Levy, A.Y., A. Rajaraman, and J.J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. in 22nd VLDB. 1996. Bombay, India.

[16] Lopes, S., J.-M. Petit, and F. Toumani, Discovering interesting inclusion dependencies: application to logical database tuning. Information Systems, 2002. 27(1): p. 1-19.

[17] Madhavan, J., et al. Corpus-based schema matching. in 21st International Conference on Data Engineering. 2005.

[18] Marchi, F.D., S. Lopes, and J.-M. Petit. Efficient Algorithms for Mining Inclusion Dependencies. in Int. Conf. on Extending Database Technology (EDBT). 2002.

[19] Naumann, F., U. Leser, and J.C. Freytag. Quality-driven Integration of Heterogeneous Information Systems. in 25th VLDB. 1999. Edinburgh, UK.

[20] Rahm, E. and P.A. Bernstein, A survey of approaches to automatic schema matching. The VLDB Journal, 2001. 10(4): p. 334-350.

[21] Tagari, M., et al., E-MSD: improving data deposition and structure quality. Nucleic Acids Res, 2006. 34(Database issue): p. D287-90.

[22] Ullman, J.D. Information Integration using Logical Views. in 6th Int. Conference on Database Theory. 1997. Delphi, Greece.