sPLMap: A probabilistic approach to schema matching

Henrik Nottelmann[1] and Umberto Straccia[2]

[1] Institute of Informatics and Interactive Systems, University of Duisburg-Essen, 47048 Duisburg, Germany, nottelmann@uni-duisburg.de
[2] ISTI-CNR, Via G. Moruzzi 1, 56124 Pisa, Italy, straccia@isti.cnr.it

Abstract. This paper introduces the first formal framework for learning mappings between heterogeneous schemas which is based on logics and probability theory. This task, also called "schema matching", is a crucial step in integrating heterogeneous collections. As schemas may have different granularities, and as schema attributes do not always match precisely, a general-purpose schema mapping approach requires support for uncertain mappings, and mappings have to be learned automatically. The framework combines different classifiers for finding suitable mapping candidates (together with their weights), and selects that set of mapping rules which is the most likely one. Finally, the framework with different variants has been evaluated on two different data sets.

1 Introduction

Federated digital libraries integrate a large number of legacy libraries and give users the impression of one coherent, homogeneous library. These libraries use different schemas (called source schemas). As users cannot deal efficiently with this semantic heterogeneity, they only see one system-wide or personalized target (or global) schema, which is defined ontologically and independent from the libraries. Then, queries are transformed from the target (global) schema into the source schemas, and documents vice versa (which is out of the scope of this paper).

Our framework sPLMap (probabilistic, logic-based mapping between schemas) combines logics with probability theory describing schema mappings. In contrast to most of the approaches available so far, this allows dealing with schemas of different granularity. If the target schema contains the two attributes "author" and "editor", and the source schema only the more general attribute "creator", this source attribute cannot be mapped onto "author" precisely but only with a specific probability. Systems with purely deterministic mappings fail in such settings.

Here, we focus on learning these schemas using documents in both schemas, but not necessarily the same documents. As a by-product, we also compute a theoretically founded measurement for the quality of a mapping.

For schemas, we adopt the document model presented in [7] with only slight modifications. Like in database systems, data types with comparison operators are explicitly modelled. However, vagueness of query formulations is one of the key concepts of Information Retrieval. Thus, it is crucial that comparison operators have a probabilistic interpretation. Vagueness is required e.g. when a user is uncertain about the exact publication year of a document or the spelling of an author name. These comparison

operators are often called “vague predicates”, we will use the term “operator” later to avoid confusion with logical predicates. For a specific attribute value the vague predicate yields an estimate of the probability that the condition is fulfilled from the user’s point of view — instead of a Boolean value as in DB systems. The schema mapping rules also cover the problem of converting one query condition, a triple of attribute name, operator and comparison value, in another schema, where potentially also the operator or the comparison value has to be modified.

The paper is structured as follows: The next section introduces a formal framework for schema mapping, based on a special probabilistic logic. Section 3 presents a theoretically founded approach for learning these schema mappings which combines the results of different classifiers. This approach is evaluated on two different test beds in section 4. Then, section 5 describes how this work is related to other approaches. The last section summarizes this paper and gives an outlook over future work.

2 Formal framework for schema mapping

This section introduces sPLMap, a formal, logics-based framework for schema mapping. It shares a lot of ideas from other approaches, e.g. [5], but is different as it is the first one which also takes data types, predicates and query mapping into consideration. It is also the first framework which is able to cope with the intrinsic uncertainty of the mapping process. The framework is based on probabilistic Datalog [8].

2.1 Probabilistic Datalog

Probabilistic Datalog (pDatalog for short) is an extension to Datalog, a variant of predicate logic based on function-free Horn clauses. Negation is allowed, but its use is limited to achieve a correct and complete model (negation is not required in this paper anyway). In pDatalog every fact or rule has a probabilistic weight $0 < \alpha \leq 1$ attached, prefixed to the fact or rule:

$$\alpha\, A \leftarrow B_1, \ldots, B_n\ .$$

Here, A denotes an atom (in the rule head), and $B_1, \ldots, B_n$ ($n \geq 0$) are atoms (the sub goals of the rule body). A weight $\alpha = 1$ can be omitted. Each fact and rule can only appear once in the program, to avoid inconsistencies. The intended meaning of a rule αr is that “the probability that any instantiation of rule r is true is α”. The following example pDatalog program expresses the fact that a person is with probability of 50% male:

$$\begin{aligned} \texttt{person(mary)} &\leftarrow \\ 0.8\,\texttt{person(ed)} &\leftarrow \\ 0.5\,\texttt{male(X)} &\leftarrow \texttt{person(X)} \end{aligned}$$

Thus, $Pr(\texttt{male(mary)}) = 0.5$, and $Pr(\texttt{male(ed)}) = 0.8 \times 0.5 = 0.4$. Formally, an interpretation structure (w. r. t. the Herbrand universe) in pDatalog is a tuple $\mathscr{I} = (\mathcal{W}, \mu)$. Here, $\mathcal{W}$ denotes a possible world (the instantiation of a the deterministic part of a pDatalog program plus a subset of the probabilistic part, where all probabilities are removed in the latter), and μ is a probability distribution over $\mathcal{W}$. An interpretation is a

tuple $I = (\mathscr{I}, w)$ such that $w \in \mathcal{W}$. The notion if truth w. r. t. an interpretation and a possible world can be defined recursively:

$$(\mathscr{I}, w) \models A \text{ iff } A \in w \,,$$
$$(\mathscr{I}, w) \models A \leftarrow B_1, \ldots, B_n \text{ iff } (\mathscr{I}, w) \models B_1, \ldots, B_n \Rightarrow (\mathscr{I}, w) \models A \,,$$
$$(\mathscr{I}, w) \models \alpha r \text{ iff } \mu(\{w' \in \mathcal{W} \colon (\mathscr{I}, w') \models r\}) = \alpha \,.$$

An interpretation is a model of a pDatalog program iff it entails every fact and rule. Given an n-ary atom A for predicate $\bar{A}$ and an interpretation $I = (\mathscr{I}, w)$, the instantiation A^I of A w. r. t. the interpretation A is defined by all $\alpha\bar{A}(c_1, \ldots, c_n)$ with $I \models \alpha\bar{A}(c_1, \ldots, c_n)$. With abuse of notation, we typically consider a relation instance as a set of probabilistically weighted tuples (the arguments of the ground facts), and do not distinguish between a relation R (an n-ary predicate) and the relation instance R^I.

2.2 Data types

We first assume a finite set **D** of elementary data types. The domain $dom(d)$ for a data type $d \in \mathbf{D}$ defines the set of possible values for d. Examples are `Text` (for English text), `Name` (person names, e.g. "John Doe"), `Year` (four digit year numbers, e.g. "2004") or `DateISO8601` for the ISO 8601 format of dates (e.g. "2004-12-31"). We further use a set **O** of operators (sometimes also called "data type predicates"). An operator is a binary relation $o \subseteq dom(d_1(o)) \times dom(d_2(o))$ defined on two data types $d_1(o), d_2(o) \in \mathbf{D}$, e.g. `contains` for text (searching for stemmed terms), $>$ or $=$ for years, or `sounds-like` for names. The operator relations have a probabilistic interpretation (which is the probability that the first value matches the second one) for supporting vague queries. In our scenario, **D** contains the data type `DOCID` (the set of all document ids); only the identity operator $\mathtt{id}_{\mathtt{DOCID}}$ is defined on it.

As we want to use variables for operators, we use a bijective mapping between operators $o \in \mathbf{O}$ and new constants $\hat{o} \in \widehat{\mathbf{O}}$ for a set of constants $\widehat{\mathbf{O}}$. Then, these operators are combined in a ternary predicate `op`:

$$\mathtt{op} = \bigcup_{o \in \mathbf{O}} \{\hat{o}\} \times o \,.$$

Again, we do not explicitly distinguish between the operators o and their constants $\hat{o}$, and use the former notation for both of them. In addition, we use a predicate `conv` for value conversion between operators:

$$\mathtt{conv}^I \subseteq \bigcup_{o_1, o_2 \in \mathbf{O}} \{\hat{o}_1\} \times dom(d_1(o_1)) \times dom(d_2(o_1)) \times \{\hat{o}_2\} \times dom(d_1(o_2)) \times dom(d_2(o_2)) \,.$$

The informal meaning of $\mathtt{conv}(O, X, Y, O', X', Y')$ is that $\mathtt{op}(O, X, Y)$ can be transformed into $\mathtt{op}(O', X', Y')$. Also `conv` can be uncertain, where the weight denotes the probability that this is a correct conversion. For example, `conv` may contain the tuples for the data types `Year2` (2-digit year numbers), `Year4` (4-digit year numbers), *FirstName* (only first names) and `Name` (complete names):

$$(\mathtt{id}_{\mathtt{Year2}}, \text{“}\mathtt{04}\text{”}, \text{“}\mathtt{04}\text{”}, \mathtt{id}_{\mathtt{Year4}}, \text{“}\mathtt{2004}\text{”}, \text{“}\mathtt{2004}\text{”}) \,,$$
$$(\geq_{\mathtt{Year2}}, \text{“}\mathtt{04}\text{”}, \text{“}\mathtt{06}\text{”}, >_{\mathtt{Year4}}, \text{“}\mathtt{2005}\text{”}, \text{“}\mathtt{2005}\text{”}) \,,$$
$$(\mathtt{id}_{\mathtt{FirstName}}, \text{“}\mathtt{John}\text{”}, \text{“}\mathtt{John}\text{”}, \mathtt{id}_{\mathtt{Name}}, \text{“}\mathtt{John\ Doe}\text{”}, \text{“}\mathtt{John\ Doe}\text{”}) \text{ with probability} < 1 \,.$$

2.3 Schemas and schema mappings

A schema $\mathbf{R} = \langle R_1, \ldots, R_n \rangle$ consists of a non-empty finite tuple of binary relation symbols. Each relation symbol R_i has a data type $d_{R_i} \in \mathbf{D}$. Then, for a (potentially uncertain) interpretation I, a schema instance is a tuple $\mathbf{R}^I = \langle R_1^I, \ldots, R_n^I \rangle$, where each relation symbol R_i is mapped onto a relation instance with the correct data types:

$$R_i \subseteq \texttt{DOCID} \times dom(d_{R_i}) \ .$$

Informally, this is the relational model of linear schemas with multi-valued schema attributes. Each attribute is modelled as a binary relation, which stores pairs of a document id and a value for that attribute.

We use the following two schemas throughout this presentation:

$$\mathbf{T} = \langle \texttt{creator}, \texttt{date} \rangle \ , \ d_{\texttt{creator}} = \texttt{Name} \ , \ d_{\texttt{date}} = \texttt{DateISO8601} \ ,$$
$$\mathbf{S} = \langle \texttt{author}, \texttt{editor}, \texttt{created} \rangle \ , \ d_{\texttt{author}} = d_{\texttt{editor}} = \texttt{Name} \ , \ d_{\texttt{date}} = \texttt{DateEnglish} \ .$$

The following example documents are used for explaining the schema matching algorithm:

$$\mathbf{T}^J := \{\texttt{creator}(\texttt{d}, ''\texttt{Miller}''), \ \texttt{creator}(\texttt{d}, ''\texttt{Smith}''), \ \texttt{date}(\texttt{d}, ''2004-12-31'')\},$$
$$\mathbf{S}^I := \{\texttt{author}(\texttt{d}, ''\texttt{Miller}''), \ \texttt{editor}(\texttt{d}, ''\texttt{Smith}''), \ \texttt{date}(\texttt{d}, ''\texttt{Dec31,2004}'')\}.$$

Schema mappings follow the GLaV approach [6]: A mapping is a tuple $\mathcal{M} = (\mathbf{T}, \mathbf{S}, \Sigma)$, where $\mathbf{T}$ denotes the target (global) schema and $\mathbf{S}$ the source (local) schema with no relation symbol in common, and Σ is a finite set of mapping constraints (pDatalog rules) of one of the forms (T_j and S_i are target and source attributes, respectively):

$$\alpha_{j,i} \ T_j(D,X) \leftarrow S_i(D,X_1), \texttt{conv}(id_{d_{T_j}}, X, X, id_{d_{S_i}}, X_1, X_1)$$
$$\texttt{op}(O,X,V) \leftarrow \texttt{conv}(O,X,V,O_1,X_1,V_1), \texttt{op}(O_1,X_1,V_1) \ .$$

For simplicity of representation, we drop the `conv` literal in the remainder of this paper.

In our example, `creator` subsumes both authors and editors, thus we have these mapping rules:

$$\texttt{creator}(\texttt{D},\texttt{V}) \leftarrow \texttt{author}(\texttt{D},\texttt{V}) \ ,$$
$$\texttt{creator}(\texttt{D},\texttt{V}) \leftarrow \texttt{editor}(\texttt{D},\texttt{V}) \ ,$$
$$\texttt{date}(\texttt{D},\texttt{V}) \leftarrow \texttt{date}(\texttt{D},\texttt{V}) \ .$$

For a schema mapping instance of a mapping $\mathcal{M} = (\mathbf{T}, \mathbf{S}, \Sigma)$ and a fixed interpretation I for $\mathbf{S}$, an interpretation J for $\mathbf{T}$ is a solution for I under $\mathcal{M}$ if and only if $\langle J, I \rangle$ (the combined interpretation over $\mathbf{T}$ and $\mathbf{S}$) satisfies Σ. The minimum solution is denoted by $J(I,\Sigma)$, the corresponding relation instance with $\mathbf{T}(I,\Sigma)$ (which is also called a minimum solution). Using pDatalog rules, the minimum solution $\mathbf{T}(I,\Sigma)$ is exactly the result of applying the rules Σ onto the instance $\mathbf{S}^I$. In our example, we have $\mathbf{T}^J = \mathbf{T}(I,\Sigma)$.

3 Learning schema mappings

This paper only deals with learning schema mappings, i.e. finding associations between attributes. The assumption is that a set of data types **D** and a set of operators **O** with the corresponding relations `op` and `conv` are both already given. Learning schema mapping in sPLMap consists of four steps: (i) we guess a potential schema mapping, i.e. a set of rules Σ_k of the form $T_j(x) \leftarrow S_i(x)$ (rules without weights yet); (ii) we estimate the quality of the mapping Σ_k; (iii) among all possible sets Σ_k, we select the "best" schema mapping according to our quality measure; and finally (iv) the weights α for rules in the selected schema mapping have to be estimated.

3.1 Estimating the quality of a schema mapping

For two schemas $\mathbf{T} = \langle T_1, \ldots, T_t \rangle$ and $\mathbf{S} = \langle S_1, \ldots, S_s \rangle$ and two interpretations I for **S** and J for **T**, the goal is to find a suitable set Σ of mapping constraints. In many cases, there is no correspondence between the tuples in both instances, so that no non-trivial mapping $\Sigma \supset \emptyset$ exists. Thus, the goal is to find the "best" set of mapping constraints Σ which maximizes the probability $Pr(\Sigma, J, I)$ that the tuples in the minimum solution $\mathbf{T}(I, \Sigma)$ under $\mathcal{M} = (\mathbf{T}, \mathbf{S}, \Sigma)$ and the tuples in **T** are plausible. Here, $\mathbf{T}(I, \Sigma)$ denotes a schema instance, and $T_j(I, \Sigma)$ the instance of relation T_j formed by the minimum solution. The set Σ can be partitioned into sets Σ_j with common head T_j, whose minimum solutions $T_j(I, \Sigma_j)$ only contain tuples for T_j:

$$\begin{aligned}
\Sigma_1 &= \{\texttt{creator(D,V)} \leftarrow \texttt{author(D,V)}\ , \texttt{creator(D,V)} \leftarrow \texttt{editor(D,V)}\}\ , \\
T_1(I, \Sigma_1) &= \{\texttt{creator(d,}''\texttt{Miller}''\texttt{)},\ \texttt{creator(d,}''\texttt{Smith}''\texttt{)}\}\ , \\
\Sigma_2 &= \{\texttt{date(D,V)} \leftarrow \texttt{date(D,V)}\}\ , \\
T_2(I, \Sigma_2) &= \{\texttt{date(d,}''2004-12-31''\texttt{)}\}\ .
\end{aligned}$$

As a consequence, each target relation can be considered independently:

$$Pr(\Sigma, J, I) = \prod_{j=1}^{t} Pr(\Sigma_j, J, I)\ .$$

The instances $T_j(I, \Sigma_j)$ and T_j are plausible if the tuples in $T_j(I, \Sigma_j)$ are plausible values for T_j, and vice versa. Using Bayes' theory, $Pr(\Sigma_j, J, I)$ can be computed as:

$$\begin{aligned}
Pr(\Sigma_j, J, I) &= Pr(T_j | T_j(I, \Sigma_j)) \cdot Pr(T_j(I, \Sigma_j) | T_j) \\
&= Pr(T_j(I, \Sigma_j) | T_j)^2 \cdot \frac{Pr(T_j)}{Pr(T_j(I, \Sigma_j))} \\
&= Pr(T_j(I, \Sigma_j) | T_j)^2 \cdot \frac{|T_j|}{|T_j(I, \Sigma_j)|}\ .
\end{aligned}$$

As building blocks of Σ_j, we use the sets $\Sigma_{j,i}$ containing only one candidate rule $\alpha_{j,i}\ T_j(D, X) \leftarrow S_i(D, X)$:

$$\begin{aligned}
\Sigma_{1,1} &= \{\texttt{creator(D,V)} \leftarrow \texttt{author(D,V)}\} & \Sigma_{2,1} &= \{\texttt{date(D,V)} \leftarrow \texttt{author(D,V)}\} \\
\Sigma_{1,2} &= \{\texttt{creator(D,V)} \leftarrow \texttt{editor(D,V)}\} & \Sigma_{2,2} &= \{\texttt{date(D,V)} \leftarrow \texttt{editor(D,V)}\} \\
\Sigma_{1,3} &= \{\texttt{creator(D,V)} \leftarrow \texttt{date(D,V)}\} & \Sigma_{2,3} &= \{\texttt{date(D,V)} \leftarrow \texttt{date(D,V)}\}\ .
\end{aligned}$$

For s source relations and a fixed j, there are also s possible sets $\Sigma_{j,i}$, and 2^s-1 non-empty combinations (unions) of them, forming all possible non-trivial sets Σ_j. To simplify the notation, we set $S_i := T_j(I,\Sigma_{j,i})$ for the instance derived by applying a single rule. For computational simplification, we assume that S_{i_1} and S_{i_2} are disjoint for $i_1 \neq i_2$. If Σ_j is formed by the r single rule sets $\Sigma_{j,i_1},\ldots,\Sigma_{j,i_r}$, then we obtain:

$$Pr(T_j(I,\Sigma_j)|T_j) = \sum_{k=1}^{r} Pr(S_{i_k}|T_j) \ .$$

Thus, the probability $Pr(\Sigma,J,I)$ can be derived from the $O(s\cdot t)$ probabilities $Pr(S_i|T_j)$. Note, however, that this is only a trick for estimating the former probability. The final output, the rule weights, use the "inverse direction", i.e. $\alpha = Pr(T_j|S_i)$. Section 3.4 shows how this rule probability is computed.

3.2 Estimating the probability that a mapping rule is plausible

Similar to LSD [3], the probability $Pr(S_i|T_j)$ is estimated by combining different classifiers $CL_1,\ldots CL_n$. Each classifier CL_k computes a weight $w(S_i,T_j,CL_k)$, which has to be normalized and transformed into $Pr(S_i|T_j,CL_k) = f(w(S_i,T_j,CL_k))$, the classifier's approximation of $Pr(S_i|T_j)$. We employ different normalization functions f:

$$\begin{aligned}
w(S_i,T_j,CL_k) &\mapsto Pr(S_i|T_j) \ , \\
f_{id}(x) &:= x \ , \\
f_{sum}(x) &:= \frac{x}{\sum_{i'} w(S_{i'},T_j,C_k)} \ , \\
f_{lin}(x) &:= c_0 + c_1 \cdot x \ , \\
f_{log}(x) &:= \frac{\exp(b_0+b_1\cdot x)}{1+\exp(b_0+b_1\cdot x)} \ .
\end{aligned}$$

The functions f_{id}, f_{sum} and the logistic function f_{log} return values in $[0,1]$. For the linear function, results below zero have to mapped onto zero, and results above one have to be mapped onto one. The function f_{sum} ensures that each value is in $[0,1]$, and that the sum equals 1. Its biggest advantage is that is does not use external parameters. In contrast, the parameters of the linear and logistic function have to be learned by regression in a system-training phase. This phase is only required once, and their results can be used for learning arbitrary many schema mappings. Of course, normalization functions can be combined. Often it is useful to bring the classifier weights in the same range (using f_{sum}), and then to apply another normalization function with parameters (e.g. the logistic function).

For the final probability $Pr(S_i|T_j,CL_k)$, we have the constraint

$$0 \leq Pr(S_i|T_j,CL_k) \leq \frac{\min(|S_i|,|T_j|)}{|T_j|} = \min(\frac{|S_i|}{|T_j|},1) \ . \qquad (1)$$

Thus, the normalized value (which is in $[0,1]$) is multiplied with $\min(|S_i|/|T_j|,1)$ in a second normalization step.

The final predictions $Pr(S_i|T_j,CL_k)$ are then combined using the Total Probability Theorem, which results in a weighted sum:

$$Pr(S_i|T_j) \approx \sum_{k=1}^{n} Pr(S_i|T_j,CL_k) \cdot Pr(CL_k) \ . \tag{2}$$

The probability $Pr(CL_k)$ describes the probability that we rely on the judgment of classifier CL_k, which can for example be expressed by the confidence we have in that classifier. We simply use $Pr(CL_k) = \frac{1}{n}$ for $1 \leq k \leq n$, i.e. the predictions are averaged.

3.3 Classifiers

Most classifiers require instances of both schemas. However, these instances do not need to describe the same objects. The instances should either be a complete collection, or a representative sample of it, e.g. acquired by query-based sampling [1]. Below, see a list of classifiers we considered.

Same attribute names. This binary classifier CL_N returns a weight of 1 if and only if the two attributes have the same name, and 0 otherwise:

$$w(S_i,T_j,CL_N) := \begin{cases} 1 & , \quad S_i = T_j, \\ 0 & , \quad \text{otherwise} \end{cases}$$

Exact tuples. This classifier CL_E (for testing and evaluation) measures the fraction of the tuples in T_j which also occur in $S_i = T_j(I,\Sigma_{j,i})$:

$$w(S_i,T_j,CL_E) := \frac{|S_i \cap T_j|}{|T_j|} \ .$$

Correct literals. This classifier CL_L (suitable in particular for numbers, URLs and other facts) measures the fraction of the tuples in T_j where the data value (the second argument, without the document id) also occurs in any tuple in S_i:

$$w(S_i,T_j,CL_L) := \frac{|\{T_j(t_1,t_2) \colon T_j(t_1,t_2) \in T_j, \exists T_j(s_1,s_2) \in S_i = T_j(I,\Sigma_{j,i}).s_2 = t_2\}|}{|T_j|} \ .$$

kNN classifier. A popular classifier for text and facts is kNN [15]. For CL_{kNN}, each attribute acts as a category, and training sets are formed for every tuple in S_l:

$$Train = \bigcup_{l=1}^{s} \{(S_l,t') \colon t' \in S_l\} \ .$$

A probabilistic variant of the scalar product is used for computing the similarity values. The values t and t' are considered as bags of words, and $Pr(w|S_i)$ and $Pr(w|T_j)$ are computed as the normalized frequencies of the words in the instances:

$$RSV(t,t') = \sum_{w \in t \cap t'} Pr(w|S_i) \cdot Pr(w|T_j) \ .$$

Naive Bayes text classifier. The classifier CL_B uses a naive Bayes text classifier [15] for text content. Again, each attribute acts as a category, and attribute values are considered as bags of words (with normalized word frequencies as probability estimations). The final formula is:

$$w(S_i, T_j, CL_B) = Pr(S_i) \cdot \sum_{x \in T_j} \prod_{w \in x} Pr(w|S_i) \ .$$

3.4 Estimating the weight of a rule

After a schema mapping (a set of rules) is learned, the weights $Pr(T_j|S_i)$ for these rules have to be computed. The probability $Pr(S_i|T_j)$ has already been computed for the quality estimation and, thus, can easily be transformed in the rule weight using Bayes theory:

$$Pr(T_j|S_i) = Pr(S_i|T_j) \cdot \frac{Pr(T_j)}{Pr(S_i)} = Pr(S_i|T_j) \cdot \frac{|T_j|}{|S_i|} \ . \tag{3}$$

As the final normalization step in section 3.2 ensures that $Pr(S_i|T_j) \leq \min(|S_i|/|T_j|, 1)$ (see equation (1)), the resulting value $Pr(T_j|S_i)$ is always in $[0, 1]$.

This completes the schema mapping learning process.

4 Experiments for learning schema mappings

This chapter describes the experiments conducted so far for evaluating sPLMap.

4.1 Evaluation setup

This section describes the test sets (source and target instances) and the classifiers used for the experiments. It also introduces different effectiveness measurements for evaluating the learned schema mappings (error, precision, recall). Experiments were performed on two different test beds [3]:

- BIBDB contains over 3,000 BibTeX entries about information retrieval and related areas. The documents are available both in BibTeX (source schema) and in a manually created standard schema (from the MIND project), derived from BibTeX via simple rules. Both schemas share a large amount of common attribute names.
- LOC is an Open Archive collection of the Library of Congress with about 1,700 documents, available in MARC 21 (source schema) and in Dublin Core (target schema). MARC 21 has a higher granularity as DC, thus a lot of DC attribute values are the concatenation of several MARC 21 attributes. Both schemas use a completely different name scheme, thus they do not have attribute names in common.

Each collection is split randomly into four sub-collections of approximately the same size. The first sub-collection is always used for learning the parameters of the normalization functions (same documents in both schemas). The second sub-collection is used

[3] `http://faure.isti.cnr.it/~straccia/download/TestBeds/ecir05-exp.tar.gz`

as source instance for learning the rules, and the third sub-collection is used as the target instance. Finally, the fourth sub-collection is employed for evaluating the learned rules (for both instances, i.e. we evaluate on parallel corpora).

Each of classifiers introduced in section 3.3 are used alone, plus the combinations $CL_{kNN} + CL_B + CL_L$ and $CL_{kNN} + CL_B + CL_L + CL_N$. The three normalization functions from section 3.2 (f_{sum}, f_{minmax} and f_{id}) are used; in every experiment, every classifier used the same normalization function.

The probability of a tuple t in the given target instance $\mathbf{T}_j^J$ is denoted by $Pr(T_j(d,v) \in T_j^J)$. Often the target instance only contains deterministic data, then we have $Pr(T_j(d,v) \in T_j^J) \in \{0,1\}$. Similarly, $Pr(T_j(d,v) \in T_j(I,\Sigma_j)) \in [0,1]$ denotes the probability of tuple t w. r. t. the minimal solution of the given source instance and the learned schema mapping, i.e. by applying the schema mapping on the source instance. Rule application includes mapping the resulting tuple weights onto 0 or 1, respectively, in the case where a rule weight α outside $[0,1]$ (due to a wrong estimation) leads to a tuple weight which is less than zero or higher than one.

The error of the mapping is defined by:

$$E(\mathcal{M}) = \frac{1}{\sum_j |U_j|} \sum_j \sum_{T_j(d,v) \in U_j} (Pr(T_j(d,v) \in T_j^J) - Pr(T_j(d,v) \in T_j(I,\Sigma_j)))^2 ,$$
$$U_j = T_j \cup T_j(I,\Sigma_j) .$$

Here, the set U_j contains the union of the given target instance tuples and the tuples created by applying the mapping rules. For each of these tuples, the squared difference of the given weight $Pr(t|T_j)$ in the target instance and the computed weight $Pr(t|T_j(I,\Sigma_j))$ is computed. Furthermore, we evaluated if the learning approach computes the correct rules (neglecting the corresponding rule weights). Similar to the area of document retrieval, precision defines how many learned rules are correct, and recall defines how many correct rules are learned. Finally, the F-measure denotes the harmonic mean of precision and recall. So, let R_L denote the set of rules (without weights) returned by the learning algorithm, and R_A the set of rules (again without weights) which are the actual ones. Then

$$precision := \frac{|R_L \cap R_A|}{|R_L|}, \quad recall := \frac{|R_L \cap R_A|}{|R_A|}, \quad F = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} .$$

4.2 Results

In the experiments presented in this section, the learning steps are as follows:

1. Find the best schema mapping
   (a) Estimate the plausibility probabilities $Pr(S_i|T_j)$ for every $S_I \in \mathbf{S}$, $T_j \in \mathbf{T}$ using the classifiers.
   (b) For every target relation T_j and for every non-empty subset of the 10 best[4] schema mapping rules having T_j as head, estimate the probability $Pr(\Sigma_j, J, I)$.

[4] These are the rules with the highest prediction $Pr(S_i|T_j)$.

(c) Select the rule set Σ_j which maximizes the probability $Pr(\Sigma_j, J, I)$.

2. Estimate the weights $Pr(T_j|S_i)$ for the learned rules by converting $Pr(S_i|T_j)$, using equation (3).
3. Compute the error, precision and recall as described above.

The results depicted in the tables 1 and 2 show that the LOC collection is much harder, as the schemas have different granularities, and both schemas do not have any attribute name in common. The error for the BIBDB collection can be quite low (below 0.1 for CL_L), while the error is in all but two cases above 0.5 for LOC. Precision is high for both collections, but higher for BIBDB. As the learner CL_N cannot learn any rule for LOC (as both schemas use completely different attribute names), the precision is not defined. For the BIBDB collection, recall can be quite high (over 0.9 for CL_{kNN} and the combined classifiers). For LOC, however, the best recall achieved is 0.3171

Averaged on both collections and all normalization functions, the error is minimized by CL_{kNN} with an error of 0.2864, followed by the two combinations with an error of 0.4334, followed by $CL_{kNN} + CL_B + CL_L$ and and $CL_{kNN} + CL_B + CL_L + CL_N$ (each 15-18% worse). Not surprisingly, CL_N and CL_E performed worst (more than 100% worse than CL_{kNN}). These results are replicated considering recall. Interestingly, CL_E yields the highest precision with 0.9250, followed by CL_L (about 14% worse) and $CL_{kNN} + CL_B + CL_L + CL_N$ (about 23% worse). The worst precision (<=0.5 on average) is obtained by CL_N and CL_B. This last result is due to the fact that CL_N does not work on the LOC collection (with no attribute names in common), but perfectly works on the BIBDB collection; while CL_B performs worst for both collection. Overall, combining classifiers can reduce the error and increase recall and precision.

Averaged on both collections and all classifiers, the best normalization functions w. r. t. the error are $f_{log} \circ f_{sum}$ (0.3331) and $f_{lin} \circ f_{sum}$ (about 25% worse). Precision is maximized for f_{id} (0.7346), while recall is maximized for $f_{log} \circ f_{sum}$ and f_{id} (both about 0.45). The experiments show that using the trivial normalization function f_{id} dramatically increases the error (70%), but performs best w. r. t. precision and recall. In other words, the trivial normalization function helps in finding the correct rules, but fails in finding good rule weights (for which a different normalization function has to be applied).

The best classifier/normalization function combination is CL_{kNN} with $f_{log} \circ f_{sum}$ with an error of 0.1261. Best precision is obtained for using CL_E with any normalization function (virtually no difference on average). Recall is maximized for $CL_{kNN} + CL_B + CL_L + CL_N$ with f_{id} (surprisingly), followed by the other normalization functions for CL_{kNN}.

As an illustrative example, in one of BIBDB runs, these two rules are returns for the target attribute `booktitle`:

$$
\begin{aligned}
0.51\ \texttt{standard\_booktitle}(\texttt{D},\texttt{X}) \leftarrow\ & \texttt{BIBDB\_booktitle}(\texttt{D},\texttt{X}'),\\
& \texttt{conv}(\texttt{id}_{\texttt{Text}},\texttt{X},\texttt{X},\texttt{id}_{\texttt{Text}},\texttt{X}',\texttt{X}')\\
0.98\ \texttt{standard\_booktitle}(\texttt{D},\texttt{X}) \leftarrow\ & \texttt{BIBDB\_journal}(\texttt{D},\texttt{X}'),\\
& \texttt{conv}(\texttt{id}_{\texttt{Text}},\texttt{X},\texttt{X},\texttt{id}_{\texttt{Text}},\texttt{X}',\texttt{X}')
\end{aligned}
$$

Notice that, for instance, a query for `booktitle` is then converted into the source schema, using the above rules, by unfolding the query into two source queries (one for `booktitle`, the other for `journal`).

5 Related work

In the field of federated databases, two approaches are distinguished (see [11, 14]). In "local as view" (LaV), the source schemas are defined as views (mappings) over a fixed global schema. This makes it easy to add a new source, but query transformation has exponential time complexity. In contrast, the global schema is defined as a view over local schemas in the "global as view" (GaV) approach. Here, query transformation can be reduced to rule unfolding, but adding new sources might require to modify the global view. The GLaV approach [6] combines the advantages of both worlds. The global schema is specified ontologically and independent from the sources, the source schema models the documents returned by the source, and mappings are defined by logical rules between query expressions. We adopt the main GLaV idea of independent schemas, but use probabilistic GaV rules, and restrict the schema structure to binary relations (for attributes).

Automatically learning rules is an important problem in machine learning, e.g. for learning relationships between taxonomies or document classifications. A general approach to this pronlem (not only for schema mapping) is described in [12]. ILP (Inductive Logic Programming) is employed for learning rules, while PAC learning algorithm is used for learning the rule weights. The approach requires the same documents in both schemas ("parallel corpora"), which is infeasible in most environments. A second drawback is that it is based on exact match only.

Similar to sPLMap, the heuristic system LSD [3] for finding 1:1 matchings in XML documents uses a linear combination of the predictions of multiple base learners (classifiers). The combination weights are learned via regression on manually specified mappings between a small number of learning schemas. LSD has several extensions, e.g. iMAP [2] for complex matchings in relational databases and GLUE [4] for matching ontologies on the semantic web (which relies on joint probability distributions).

Information theory measures and graph matching is used in [10]. Graphs are constructed from the schemas, where the attributes form the nodes, labelled with the entropy of the attribute. All nodes are connected, the edges are labelled with the mutual information (correlation between two distributions). Both measures do not require any interpretation of the data, i.e. data type do not have to be considered. A distance measure is defined, and optimum graph matchings is applied for finding schema mappings.

A completely different approach is taken in MGS [9]. It aims at finding a "hidden model", a schema that probabilistically generates the observed schemas. A hidden model is a partition of the attribute space with a probability function of the partitions and their attributes. The first step finds cliques in the graph where two nodes (attributes) are connected if they are not occurring in the same schema. These cliques do not contradict the schemas. The problem of selecting those cliques which form a partitions is then reduced to a set-cover problem, and the probability functions are computed by

maximum-likelihood. In a final step, χ^2 statistical testing is employed for finding sufficiently consistent models.

6 Conclusion and outlook

Learning rules automatically is an important problem in machine learning, and a large amount of work has been devoted to it. Schema matching is one instantiation of this task, where correspondences ("rules") between two heterogeneous schemas have to be found. In this paper we introduced sPLMap, a formal GLaV-like framework for schema matching, where the mappings are defined as uncertain rules in probabilistic Datalog. These schema mapping rules do not only cover transforming data from one attribute into another, but can also be used for transforming query conditions (potentially also modifying the operator or the comparison value). Although the framework is based on logics, real-world documents and queries with a linear schema can easily be converted into the logical formalism.

The framework sPLMap also covers learning of schema mappings. Different classifiers are used for predicting the probability that tuples in a target relation are plausible for a source relation. Similar to LSD, these predictions are combined to an overall approximation of rule probability. From these probabilities, a probability that a set of such schema mapping rules is plausible is derived. Finally, the rule weights have to be computed. The evaluation shows good performance in error, precision and recall, depending on the chosen classifier(s) and normalization function(s). In particular, instance-based classifiers perform surprisingly well.

The results in this paper can be employed in different ways:

1. Specific schema mapping services can be built automatically. Each schema mapping service has associated two schemas, and it is responsible for mapping between these two schemas. The mapping should be learned automatically instead of being defined manually.
2. Peer-to-peer networks are dynamic scenarios where services can dynamically join and leave, so the system can–for each query–only consider the services which are currently available. Using a decision-theoretic model as for the narrower task of resource selection, we have to find a quality measurement for a schema mapping service.

We mainly target at the information exchange problem: Two schemas are given, and an object instance in one schema is transformed into an instance of the other schema. Our mechanism could also be used for the problem of information integration: Given two source schemas, a mediated schema of them has to be created. A solution would be to build the union of both schemas, learn mapping rules, and remove useless attributes.

In future, more variants should be developed and evaluated to improve the quality of the learning mechanism. Additional classifiers could consider the data types of two attributes, could use a thesaurus for finding synonym attribute names, or could use other measures like KL-distance or mutual information. Instead of averaging the classifier predictions, the weights could be learned via regression. Odds or statistical significance tests could be employed for determining the best schema mapping.

In this work, the `conv` predicate is given. In environments with large numbers of data types, or a dynamically changing set of data types, learning the conversion predicate would be desirable, e.g. the conversion from centimeter to inch.

A more basic extension is the application onto ontologies. Instead of linear schemas, classification hierarchies are given. The task then is to map instances from one class onto classes in the other hierarchy. We are currently developing a variant oPLMap which is able to infer mapping rules between ontologies.

7 Acknowledgements

This work is supported in part by ISTI-CNR (project "Distributed Search in the Semantic Web") and in part by the DFG (grant BIB47 DOuv 02-01, project "Pepper").

References

[1] J. Callan and M. Connell. Query-based sampling of text databases. *ACM Transactions on Information Systems*, 19(2):97–130, 2001.

[2] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: Discovering complex semantic matches between database schemas. In *SIGMOD 2004*, 2004.

[3] A. Doan, P. Domingos, and A. Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *SIGMOD Conference*, 2001.

[4] A. Doan, J. Madhavan, R. Dhamankar, P. Domingos, and A. Halevy. Learning to match ontologies on the semantic web. 2004.

[5] R. Fagin, P. G. Kolaitis, W.-C. Tan, and L. Popa. Composing schema mappings: Second-order dependencies to the rescue. In *Proceedings PODS*, 2004.

[6] M. Friedman, A. Y. Levy, and T. D. Millstein. Navigational plans for data integration. In *Proceedings of 16th Natl Conf on Artificial Intelligence*, pages 67–73, 1999.

[7] N. Fuhr. Towards data abstraction in networked information retrieval systems. *Information Processing and Management*, 35(2):101–119, 1999.

[8] N. Fuhr. Probabilistic Datalog: Implementing logical information retrieval for advanced applications. *Journal of the American Society for Information Science*, 51(2):95–110, 2000.

[9] B. He and K. C.-C. Chang. Statistical schema matching across web query interfaces. In Papakonstantinou et al. [13].

[10] J. Kang and J. F. Naughton. On schema matching with opaque column names and data values. In Papakonstantinou et al. [13].

[11] M. Lenzerini. Data integration: a theoretical perspective. In *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS-02)*, pages 233–246. ACM Press, 2002.

[12] H. Nottelmann and N. Fuhr. Learning probabilistic Datalog rules for information classification and transformation. In H. Paques, L. Liu, and D. Grossman, editors, *Proceedings of the 10th International Conference on Information and Knowledge Management*, pages 387–394, New York, 2001. ACM.

[13] Y. Papakonstantinou, A. Halevy, and Z. Ives, editors. *Proceedings SIGMOD 2003*, 2003.

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

[15] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002.

	f_{id}	f_{sum}	$f_{lin} \circ f_{sum}$	$f_{log} \circ f_{sum}$
CL_E	0.8615	0.3689	0.3689	0.3689
CL_L	0.4042	0.0855	0.0854	0.0548
CL_N	0.2639	0.2639	0.2639	0.2639
CL_{kNN}	0.1696	0.0578	0.0535	0.0382
CL_B	0.7024	0.1607	0.1621	0.1629
CL_{kNN}+CL_B+CL_L	0.3287	0.0694	0.0686	0.0555
CL_{kNN}+CL_B+CL_L+CL_N	0.3225	0.0920	0.0916	0.0806

(a) Error

	f_{id}	f_{sum}	$f_{lin} \circ f_{sum}$	$f_{log} \circ f_{sum}$
CL_E	1.0000	1.0000	1.0000	1.0000
CL_L	0.8750	0.8750	0.8750	0.8750
CL_N	1.0000	1.0000	1.0000	1.0000
CL_{kNN}	0.7692	0.7692	0.7692	0.7500
CL_B	0.5000	0.5000	0.5000	0.4667
CL_{kNN}+CL_B+CL_L	0.7692	0.5882	0.5882	0.5263
CL_{kNN}+CL_B+CL_L+CL_N	1.0000	0.7692	0.7692	0.7692

(b) Precision

	f_{id}	f_{sum}	$f_{lin} \circ f_{sum}$	$f_{log} \circ f_{sum}$
CL_E	0.3636	0.3636	0.3636	0.3636
CL_L	0.6364	0.6364	0.6364	0.6364
CL_N	0.6364	0.6364	0.6364	0.6364
CL_{kNN}	0.9091	0.9091	0.9091	0.8182
CL_B	0.5455	0.5455	0.5455	0.6364
CL_{kNN}+CL_B+CL_L	0.9091	0.9091	0.9091	0.9091
CL_{kNN}+CL_B+CL_L+CL_N	1.0000	0.9091	0.9091	0.9091

(c) Recall

	f_{id}	f_{sum}	$f_{lin} \circ f_{sum}$	$f_{log} \circ f_{sum}$
CL_E	0.5333	0.5333	0.5333	0.5333
CL_L	0.7368	0.7368	0.7368	0.7368
CL_N	0.7778	0.7778	0.7778	0.7778
CL_{kNN}	0.8333	0.8333	0.8333	0.7826
CL_B	0.5217	0.5217	0.5217	0.5385
CL_{kNN}+CL_B+CL_L	0.8333	0.7143	0.7143	0.6667
CL_{kNN}+CL_B+CL_L+CL_N	1.0000	0.8333	0.8333	0.8333

(d) F-measure

Table 1. Experimental results – BIBDB

	f_{id}	f_{sum}	$f_{lin} \circ f_{sum}$	$f_{log} \circ f_{sum}$
CL_E	0.7655	0.7602	0.7602	0.7613
CL_L	0.6754	0.7207	0.7110	0.6266
CL_N	1.0000	1.0000	1.0000	1.0000
CL_{kNN}	0.5948	0.5874	0.5763	0.2140
CL_B	0.6273	0.6315	0.5708	0.2760
CL_{kNN}+CL_B+CL_L	0.6250	0.5561	0.5527	0.3837
CL_{kNN}+CL_B+CL_L+CL_N	0.6421	0.5427	0.5545	0.3771

(a) Error

	f_{id}	f_{sum}	$f_{lin} \circ f_{sum}$	$f_{log} \circ f_{sum}$
CL_E	0.8889	0.8889	0.8889	0.7333
CL_L	0.8000	0.8000	0.8000	0.4737
CL_N	N/A	N/A	N/A	N/A
CL_{kNN}	0.7059	0.7059	0.7059	0.1688
CL_B	0.4375	0.4375	0.4375	0.1731
CL_{kNN}+CL_B+CL_L	0.7692	0.6429	0.6923	0.3000
CL_{kNN}+CL_B+CL_L+CL_N	0.7692	0.6429	0.6923	0.3000

(b) Precision

	f_{id}	f_{sum}	$f_{lin} \circ f_{sum}$	$f_{log} \circ f_{sum}$
CL_E	0.1951	0.1951	0.1951	0.2683
CL_L	0.1951	0.1951	0.1951	0.2195
CL_N	0.0000	0.0000	0.0000	0.0000
CL_{kNN}	0.2927	0.2927	0.2927	0.3171
CL_B	0.1707	0.1707	0.1707	0.2195
CL_{kNN}+CL_B+CL_L	0.2439	0.2195	0.2195	0.2195
CL_{kNN}+CL_B+CL_L+CL_N	0.2439	0.2195	0.2195	0.2195

(c) Recall

	f_{id}	f_{sum}	$f_{lin} \circ f_{sum}$	$f_{log} \circ f_{sum}$
CL_E	0.3200	0.3200	0.3200	0.3929
CL_L	0.3137	0.3137	0.3137	0.3000
CL_N	N/A	N/A	N/A	N/A
CL_{kNN}	0.4138	0.4138	0.4138	0.2203
CL_B	0.2456	0.2456	0.2456	0.1935
CL_{kNN}+CL_B+CL_L	0.3704	0.3273	0.3333	0.2535
CL_{kNN}+CL_B+CL_L+CL_N	0.3704	0.3273	0.3333	0.2535

(d) F-measure

Table 2. Experimental results – LOC