Database Schema Matching Using Machine Learning with Feature Selection

Jacob Berlin and Amihai Motro

Information and Software Engineering Department
George Mason University, Fairfax, VA 22030
{jberlin,ami}@gmu.edu

Abstract. Schema matching, the problem of finding mappings between the attributes of two semantically related database schemas, is an important aspect of many database applications such as schema integration, data warehousing, and electronic commerce. Unfortunately, schema matching remains largely a manual, labor-intensive process. Furthermore, the effort required is typically linear in the number of schemas to be matched; the next pair of schemas to match is not any easier than the previous pair. In this paper we describe a system, called Automatch, that uses machine learning techniques to automate schema matching. Based primarily on Bayesian learning, the system acquires probabilistic knowledge from examples that have been provided by domain experts. This knowledge is stored in a knowledge base called the *attribute dictionary*. When presented with a pair of new schemas that need to be matched (and their corresponding database instances), Automatch uses the attribute dictionary to find an optimal matching. We also report initial results from the Automatch project.

1 Introduction

Schema matching is the problem of finding mappings between the attributes of two semantically related database schemas. The schema matching problem is an important, current issue for many database applications such as schema integration, data warehousing, and electronic commerce [12,15]. Unfortunately, schema matching remains largely a manual, labor-intensive process. Furthermore, the effort required is typically linear in the number of schemas to be matched; the next pair of schemas to match is not any easier than the previous pair. Thus, database applications that require schema matching are limited to environments in which the set of member information sources is small and stable. These applications would scale-up to much larger communities of member sources if the schema matching "bottleneck" was broken by automating the matching process.

In this paper we discuss such a system, called Automatch, for automating the schema matching process. Based primarily on Bayesian learning, the system acquires probabilistic knowledge from examples of schemas that have been "mapped" by domain experts into a knowledge base of database attributes called the *attribute dictionary*. Roughly speaking, this dictionary characterizes different

A. Banks Pidduck et al. (Eds.): CAISE 2002, LNCS 2348, pp. 452–466, 2002.

attributes by means of their *possible values* and the *probability estimates* of these values. Furthermore, the dictionary may be extended to contain any attribute metadata that has a probabilistic interpretation (e.g. attribute names or string patterns).

When presented with a pair of "client" schemas that need to be matched (and their corresponding database instances), Automatch matches them "through" its dictionary. Using probabilistic methods, an attempt is made to match every attribute of one client schema with every attribute of the other client schema, resulting in individual "scores." An optimization process based on a Minimum Cost Maximum Flow network algorithm finds the overall optimal matching between the two client schemas, with respect to the sum of the individual attribute matching scores.

To overcome the problem of very large dictionaries caused by very large attribute domains, Automatch employs statistical *feature selection* techniques to learn an efficient representation of the examples. That is, each attribute is represented with a minimal set of most informative values. Thus the attribute dictionary is made human understandable through aggressive reduction in the number of values. Although the example schemas may contain many thousands of values, we are able to focus learning on a very small subset, consisting of as few as 10% of the initial values.

The results of our initial experimentation with Automatch are encouraging as they show performance that exceeds 70% (measured as the harmonic mean of the soundness and the completeness of the matching process). Although the attribute dictionary was built for Automatch, we conjecture that it could be employed as a knowledge asset in other schema matching systems.

The remainder of this paper is organized as follows. Section 3 describes the basic methodology of Automatch; in particular, the probabilistic information in the acquired knowledge base and how it is used to infer optimal matchings between "client" schemas. Section 4 describes alternative methods for reducing the size of the knowledge base through feature selection. Section 5 explains the experiment and its conclusions. Section 6 summarizes the contributions and suggests future research directions. We begin with a brief discussion of other published approaches and how they are related to Automatch.

2 Related Work

A thorough discussion of schema matching techniques and implementations can be found in [6,11,15]. Here we mention two such approaches and compare them to Automatch. Automated schema matching can be classified as rule based and learner based [6].

The Artemis system [5] is a rule-based approach for schema integration. This system determines the *affinity* of attributes from two schemas in a pair-wise fashion. Affinity is based on comparisons of attribute names, structure, and domain types and is scored on a [0,1] interval. The process relies on thesauri to determine semantic relationships. The system uses hierarchical clustering based on

affinity values to group together related attributes. Finally, a set of unification rules are employed to interactively guide a user through the construction of an integrated schema. In contrast with Automatch, Artemis considers schema information; Automatch considers instance information. Furthermore, knowledge in Artemis is "pre-coded" in the thesaurus and unification rules; knowledge in Automatch is learned from examples.

SemInt [9,10] is a learner-based system that uses neural networks to identify similar attributes from different schemas. This system uses a combination of schema and instance information. Schema information includes such information as data types, field length, and constraint information. Instance information includes such information as value distributions, character ratios, numeric mean and variance.

For each type of information the system exploits, it determines a numerical value on a $[0, 1]$ interval. A tuple of these numerical values for one attribute is the *signature* of the attribute. The system uses these signatures to cluster similar attributes within the same schema. The system then uses the signatures of the cluster centers to train a neural network to output an attribute category based on the input signatures. Given a new schema, the system determines the signature of each schema attribute using the same type of schema and instance information used for training. These signatures are then applied to the neural network to determine the category of the respective attributes. In contrast with Automatch, SemInt uses a fixed set of features for learning; Automatch combines feature selection with learning to find an optimal set of features for a given problem domain. Furthermore, SemInt discovers matches to attribute clusters; Automatch discovers matches to individual attributes.

3 Methodology

This section describes the basic methodology of Automatch, providing details of its data structures and algorithms. It begins with an intuitive description of the approach and a formal description of the problem.

3.1 The Overall Approach

Automatch is based on a knowledge base about schema attributes which is constructed from examples. When presented with two new "client" schemas that need to be matched (and their corresponding database instances), Automatch checks every client attribute against its attribute dictionary, obtaining individual "matching scores" for each pair of client attribute and dictionary attribute.

These client-dictionary attribute scores are combined to generate client-client attribute scores. To illustrate, assume B is an attribute of one client scheme, C is an attribute of the other client scheme, and A is an attribute of the dictionary, and assume that the matching of B to A is scored w_1 and the matching of C to A is scored w_2; then the matching $B \leftrightarrow C$ receives the score $w_1 + w_2$.[1]

[1] We combine the individual scores by their sum, but other combinations are also possible; for example, their product.

In turn, these individual client-client attribute scores are combined to generate overall schema-schema matching scores. To illustrate, assume schemas $R_1 = \{B_1, B_2\}$ and $R_2 = \{C_1, C_2\}$ and assume the client-client attribute scores: $w_1 : B_1 \leftrightarrow C_1$, $w_2 : B_1 \leftrightarrow C_2$, $w_3 : B_2 \leftrightarrow C_1$, and $w_4 : B_2 \leftrightarrow C_2$. The schema matching $\{B_1 \leftrightarrow C_2, B_2 \leftrightarrow C_1\}$ is then scored $w_2 + w_3$. Other schema matchings are scored similarly.

In a subsequent optimization process, Automatch finds the schema matching with the highest schema-schema score.

3.2 Formalization of the Problem

Our formalization is based on the relational model. However, we are confident that the methods can be extended to other models, such as the object-oriented or the semi-structured models. A *database schema* is simply a finite set of *attributes* $\{A_1, \ldots, A_n\}$. Given two database schemas $R_1 = \{B_1, \ldots, B_p\}$ and $R_2 = \{C_1, \ldots, C_q\}$, a *matching* is a mapping between a subset of R_1 and a subset of R_2.

We assume a knowledge base about database attributes, called the *attribute dictionary* and denoted D. In this knowledge base, each attribute is characterized by a select set of possible values and their probability estimates.

In addition, we assume a *scoring function* f that, given (1) the attribute dictionary D, (2) a pair of database schemas R_1 and R_2, (3) a pair of corresponding database instances r_1 and r_2, and (4) a matching between R_1 and R_2, issues a value (a real number), that indicates the "goodness" of the matching.

The problem is then to find the *best* matching for two given schemas R_1 and R_2. This abstract description leaves two major issues to be discussed in detail:

1. The nature of the attribute dictionary D and the scoring function f.
2. The optimization of f (i.e., finding the best schema matching).

These two issues are discussed in the next two subsections.

3.3 The Attribute Dictionary and the Scoring Function

The attribute dictionary D consists of a finite set of schema *attributes* $\{A_1, \ldots, A_r\}$. Each attribute in the attribute dictionary is characterized by a set of *possible values* and their *probability estimates*. The attribute dictionary serves as a knowledge base that accumulates information about attributes. All attempts to match attributes of client schemas refer to this knowledge base. We use Bayesian learning to populate the attribute dictionary with example values provided by domain experts.

Recall from the intuitive description in Section 3.1 that the first task is to determine client-dictionary attribute scores.

Let X be a client attribute, let A denote a dictionary attribute, and let V denote a set of values that are observed in X (these values are derived from the instance of the client schema to which X belongs).

Let $P(A)$ be the *prior* probability that X maps to A (before observing any values of X), let $P(V)$ represent the unconditional probability of observing values V in X, and let $P(V|A)$ represent the conditional probability of observing the values V, given that X maps to A. Bayes Theorem states that

$$P(A|V) = \frac{P(V|A) \cdot P(A)}{P(V)} \ . \tag{1}$$

$P(A|V)$ is referred to as the *posterior* probability that X maps to A, because it reflects the probability that a mapping of X to A holds *after* the values V have been observed. This posterior probability serves as the score of the client attribute X and the dictionary attribute A.

Letting V be a sequence of values $(v_1, \ldots, v_n)$, and assuming *conditional independence* of values given the mapping, the client-dictionary attribute score is

$$M(X, A) = \frac{P(A)}{P(V)} \cdot \prod_{k=1}^{n} P(v_k|A) \ . \tag{2}$$

Although the attribute values may not be conditionally independent, such an assumption has been shown to be an acceptable approach, aimed at reducing the number of probabilities to a tractable amount while not sacrificing optimality [7,8,13].

To build the attribute dictionary for each attribute A we must learn and store the probability estimates $P(A)$, $P(\neg A)$, $P(v|A)$, and $P(v|\neg A)$ for all dictionary attributes A and values v. Note that we do not need to learn $P(V)$ because this term is determined by the requirement that $M(X, A) + M(X, \neg A) = 1$.

$P(A)$, the probability that a client attribute X maps to A, is estimated by the proportion of examples provided by the domain expert that have been mapped to A. $P(v|A)$, the probability that attribute value v occurs given that a mapping to A holds, is estimated by counting the occurrences of v in the set of examples provided by the domain expert. The remaining terms are learned in a similar fashion. For numeric data values, we assume a normal distribution and use the normal probability density function to estimate the conditional probabilities. A thorough discussion of the algorithms for estimating these terms is reported in [4]. A critical selection process that reduces the number of values v that are maintained for each attribute A is discussed in Section 4.

3.4 Optimal Schema Matching

Assume now two given schemas R_1 and R_2 with their corresponding instances r_1 and r_2, and let D denote the attribute dictionary.

The scores $M(X, A)$ from Equation 2 are calculated for each attribute X in the given schema and for each attribute A of the dictionary. A threshold is then adopted, and scores that are below this threshold are interpreted as evidence that X should not be mapped to A. These results may be represented in a weighted tripartite graph in which nodes correspond to attributes, edges correspond to matches, and edge weights correspond to the posterior probabilities.

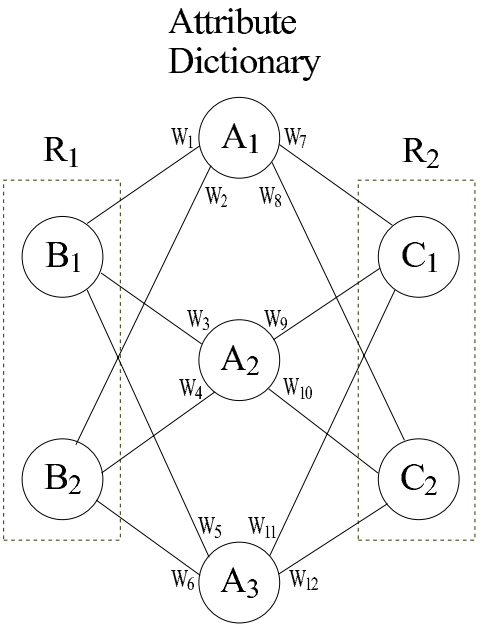

Fig. 1. Weighted tripartite graph for representing individual attribute-attribute scores

Figure 1 shows such a graph for a simple case in which $R_1 = \{B_1, B_2\}$, $R_2 = \{C_1, C_2\}$, and $D = \{A_1, A_2, A_3\}$. This example shows a *full* tripartite graph (every node in the left or right partitions is connected to every node in the center partition), but the use of a threshold implies that in general the graph need not be full.

Recall from the intuitive description in Section 3.1, that the client-dictionary attribute scores w_i are combined to generate client-client attribute scores. Note, however, that every two client attributes may be matched through every dictionary attribute. In the example, B_1 and C_1 may be matched through A_1 (with score $w_1 + w_7$), through A_2 (with score $w_3 + w_9$) and through A_3 (with score $w_5 + w_{11}$). Note that associating a dictionary attribute with every attribute match is like providing a *common type* for the matching attribute pair.

In turn, client-client attribute scores are used in generating overall schema-schema scores. In the example, the schema matching comprising of $B_1 \overset{A_1}{\leftrightarrow} C_2$ and $B_2 \overset{A_3}{\leftrightarrow} C_1$ receives the score $w_1 + w_8 + w_6 + w_{11}$. Obviously, the number of possible matchings between R_1 and R_2 is too high for a simple process that enumerates all the matchings and scores each.

One obvious approach for matching R_1 and R_2 is to choose for each client attribute the most probable dictionary attribute. For instance, in the example,

the highest of w_1, w_3 and w_5 will determine whether B_1 is mapped to A_1, A_2 or A_3. Then a mapping can be established between those schema attributes that share a node in the attribute dictionary. In the example, assume that the highest of w_1, w_3 and w_5 is w_3; (i.e., B_1 is best mapped to A_2), and assume that the highest of w_8, w_{10} and w_{12} is w_{10} (i.e., C_2 is best mapped to A_2); the conclusion would then be that B_1 is best matched with C_2. The problem with such an approach is that it easily leads to ambiguity. In the example, if the optimal mappings correspond to the edges with weights w_3, w_4, w_9 and w_{10}, we have established a match between the schemas, but the attribute mapping is ambiguous. Furthermore, the approach easily leads to no match; e.g., if the optimal mappings correspond to the edges with weights w_1, w_6, w_9, and w_{10}.

To avoid these pitfalls, we impose an additional constraint on the matching of R_1 and R_2. Specifically, we limit our search to schema mappings in which the paths between attributes in R_1 and R_2 are free of intersections. That is, two attributes of a client scheme never map to the same dictionary attribute. The resulting problem can then be solved using efficient flow network techniques. Towards this, we must first extend the tripartite graph in several ways.

First, we add two nodes to the graph: a source node S on the left, which is then connected to all the R_1 nodes, and a target node T on the right, which is then connected to all the R_2 nodes. Next, we split each attribute dictionary node A into two nodes, A^{in} and A^{out}. Each A_i^{in} is connected to its corresponding A_i^{out} node. Next, we reconnect the edges from R_1 and R_2 to the appropriate A_{in} or A_{out} node. Finally, each edge is given direction, capacity, and cost. All edges are directed away from the source node S and towards the target node T. The capacity for each edge is 1 (thus, the flow through an edge will be either 0 or 1). The cost of each of the new edges added to the graph is 0. The cost of each of the old edges is the negation of the edge weight. Figure 2 shows the new graph for the example of Figure 1. Edge capacities and costs were omitted for clarity.

The reason for the negation of the weights is that we will be using an algorithm that searches for a minimum when we actually wish to find the maximum (finding a maximum is equivalent to finding the minimum of the negation). With these modifications, we can now find a matching between the schemas R_1 and R_2 that conforms to our constraints by using a Minimum Cost Maximum Flow network algorithm [1]. In the current implementation of Automatch, we use the LEDA software package for this purpose [2].

Specifically for Figure 2, since the source has two outgoing edges of capacity 1 and the target has two incoming edges of capacity 1 (i.e., two attributes are matched on each side), the maximum flow is 2. Thus, we seek to find the edges in the graph that have the minimum cost while supporting a maximum flow of 2. The edges in this set correspond to the optimal mapping of attributes of R_1 to R_2.

Note that when the client schemas do not have the same number of attributes, some of the attributes of the larger schema will be matchless. Moreover, since the tripartite is not necessarily full, the optimal matching may leave attributes

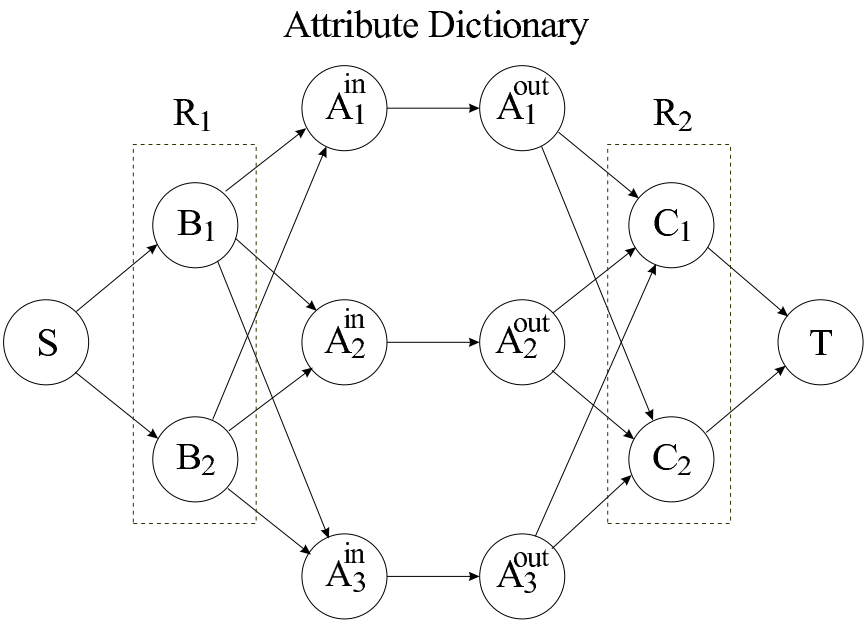

Fig. 2. Minimum-Cost-Maximum-Flow graph for finding optimal schema matching

in the *smaller* schema matchless as well. This is not an undesirable consequence, as it simply indicates that the client schemas include attributes that are unique to their schemas.

4 Optimal Selection of Dictionary Values

Recall that the attribute dictionary of Automatch represents each attribute with a set of possible values and their probability estimates. For schema attributes that contain text, the number of needed probabilities is proportional to the number of unique values of this attribute. An attribute such as *CustomerName* could assume thousands of values, thus imposing considerable space and processing requirements. Furthermore, not all of these probabilities are equally informative. Indeed, many of them are either uninformative (irrelevant) or misleading (noise).

A critical consideration in our methods is to reduce the dictionary representation of attributes while retaining the most informative values. In machine learning terminology these values are called *features* and the reduction process is called *feature selection*. To reduce the size of the Automatch dictionary, we have tested and compared three statistical feature selection strategies: Mutual Information, Information Gain, and Likelihood Ratio. The former two strategies

are commonly used for feature selection; to our knowledge the latter strategy has not been used for this purpose.

We will discuss each feature selection strategy in turn. Common to all these approaches is that each feature is assigned a "score." These feature scores can be calculated from the probability estimates in the attribute dictionary. In all of these approaches, higher scores are better. Once these scores have been calculated for a given approach, a percentage of the highest scoring features is retained with ties broken arbitrarily.

Finally, we must normalize the probabilities of the remaining features to sum to unity. Thus, statistical feature selection imposes very little overhead in our approach. In contrast, other machine learning approaches (e.g. neural networks, rule learners, etc.) must execute their respective learning algorithms after feature selection is completed.

4.1 Mutual Information

Mutual information has been used previously as a feature selection strategy in information retrieval tasks such as [16]. The mutual information of a value v and an attribute A is defined as

$$MI(v, A) = \log \frac{P(v \wedge A)}{P(v) \cdot P(A)} \ . \tag{3}$$

When v and A are independent, the mutual information of v and A is zero. Intuitively, $P(v)$ is a measure of the event that a value v occurs in the client attribute X, and $P(A)$ is a measure of the event the client attribute X is mapped to the dictionary attribute A. Hence, $MI(v, A)$ is a measure of the *co-occurrence* of these two events. For example, if the events are independent (their co-occurrence is unbiased), then the mutual information is 0.

For the purpose of characterizing dictionary attributes, we wish to retain the values that have the greatest score regardless of whether they favor A or $\neg A$. Therefore, we score values in the MI approach using this formula:

$$MI_{max}(v, A) = \max \left\{ \log \frac{P(v \wedge A)}{P(v) \cdot P(A)} \ , \ \log \frac{P(v \wedge \neg A)}{P(v) \cdot P(\neg A)} \right\} \ . \tag{4}$$

The values v with the highest $MI_{max}(v, A)$ are chosen as the characterization of attribute A. The actual number of values chosen is discussed in Section 5.2.

4.2 Information Gain

Information gain is often used in machine learning to determine the value of a particular feature [13]. Given a client attribute X and a dictionary attribute A, the issue is whether X maps to A or not. This issue may be formatted as a binary *message*: 1 if yes, 0 if no.

Denote $P(A)$ the probability that X maps to A. Assume first that our only knowledge is the proportion of attributes that are mapped to A (how "popular" A is as a target of mappings). The entropy (information content) of the message is then

$$H = -(P(A) \cdot \log P(A) + P(\neg A) \cdot \log P(\neg A)) \; . \tag{5}$$

Assume now that we know a new fact: $v \in X$. The new entropy (information content) of the message is

$$H_1 = -(P(A \mid v) \cdot \log P(A \mid v) + P(\neg A \mid v) \cdot \log P(\neg A \mid v)) \; . \tag{6}$$

Assume now that we know an alternative fact: $v \notin X$. The new entropy (information content) of the message is

$$H_2 = -(P(A \mid \neg v) \cdot \log P(A \mid \neg v) + P(\neg A \mid \neg v) \cdot \log P(\neg A \mid \neg v)) \; . \tag{7}$$

H_1 and H_2 may be combined using $P(v)$, the probability that v is in X. Then the entropy (information content) of the message is

$$H' = P(v) \cdot H_1 + p(\neg v) \cdot H_2 \; . \tag{8}$$

The *information gained* by knowing the presence or absence of v is

$$IG(v, A) = H - H' \; . \tag{9}$$

4.3 Likelihood Ratio

The likelihood ratio for a value v and attribute A, defined as $P(v|A)/P(v|\neg A)$, measures the *retrospective* support given to A by the occurrence of v [14]. The likelihood ratio produces scores on the interval $(0, \infty)$. It has a value of 1 if the feature provides no support. Likelihood ratios greater than 1 indicate that the feature supports A; likelihood ratios less than 1 indicate that the feature supports $\neg A$.

For the task at hand, we wish to retain the features that provide the most support regardless of whether they favor A or $\neg A$. The features that favor A are on the interval $(1, \infty)$, with higher values indicating stronger support, whereas the features that favor $\neg A$ are on the interval $(0, 1)$, with lower values indicating stronger support. Consequently, it is difficult to use the likelihood ratio as defined, because higher scores are not necessarily better. For this reason, we use an adjustment that inverts the likelihood ratios that support $\neg A$, placing them on the same scale as likelihood ratios that support A, and then choose the stronger of the supports:

$$LR(v, A) = \max \left\{ \frac{P(v|A)}{P(v|\neg A)}, \frac{P(v|\neg A)}{P(v|A)} \right\} \; . \tag{10}$$

This strategy produces scores on the interval $(1, \infty)$ and higher scores are always better.

5 Experimentation

5.1 Setting Up the Experiment

To experiment with the methods discussed in this paper, we built an attribute dictionary for computer retail information with the following attributes: *DesktopManufacturer, MonitorManufacturer, PrinterManufacturer, DesktopModel, MonitorModel, PrinterModel, DesktopCost, PeripheralCost, Inventory.*

Data for this experiment was taken from the web sites of 15 different computer retailers (e.g. Gateway, Outpost, etc). A total of 22 relations were extracted. The data was collected off-line from HTML web pages and imported into relational database tables accessible through the ODBC protocol.

To experiment with this data, we used a procedure from data mining called *stratified cross-validation* which we briefly describe (see [17] for a complete description). Each of the 22 schemas was manually mapped into our attribute dictionary. We then partitioned these 22 schemas into three *folds* of approximately equal size. Using two folds for learning and one fold for testing, we repeated the experiment for the three possible combinations of folds. For the test fold, we chose two schemas at a time (for all possible combinations) and used Automatch to match the schemas. We used the manually constructed mappings to judge the mappings which Automatch concluded.

5.2 Measuring Performance

To measure performance, each schema-matching result was interpreted as set of mapping decisions for pairs of schema attributes $\langle R_1(B_i), R_2(C_j)\rangle$, where i ranges over all the attributes of R_1 and j ranges over all the attributes of R_2. Each of these attribute mapping decisions falls into one of four sets, A, B, C, and D, where

A = True Positives (decision to map $R_1(B_i)$ to $R_2(C_j)$ is correct).
B = False Negatives (decision to not map $R_1(B_i)$ to $R_2(C_j)$ is incorrect).
C = False Positives (decision to map $R_1(B_i)$ to $R_2(C_j)$ is incorrect).
D = True Negatives (decision to not map $R_1(B_i)$ to $R_2(C_j)$ is correct).

The ratio $|A|/(|A| + |C|)$ is the proportion of true positives among the cases thought to be positive; i.e., it measures the accuracy of Automatch when it decides *True*. The ratio $|A|/(|A| + |B|)$ is the proportion of positives detected by Automatch among the complete set of positives; i.e., it measures the ability to detect positives. Specifically to our application, the former ratio measures the *soundness* of the discovery process, and the latter ratio measures its *completeness*. These two ratios are known from the field of information retrieval as *precision* and *recall*, but we shall refer to them here as the soundness and completeness of the schema matching process.

To simplify the comparison of the three feature selection approaches, we combined soundness and completeness into a single performance measure using their *harmonic mean*. The harmonic mean of precision and recall is often used

in information retrieval whenever a single performance measure is preferred [3]. The harmonic mean for our mapping problem is calculated as

$$F(x) = 2 \cdot \frac{S(x) \cdot C(x)}{S(x) + C(x)} \tag{11}$$

where $S(x)$ and $C(x)$ are the soundness and completeness of the discovery process at a given percent reduction x in the feature space. The harmonic mean assumes high values only when both soundness and completeness are high. Thus, maximizing the harmonic mean can be thought of as the best compromise between soundness and completeness.

To measure the performance of each of the feature selection strategies that were discussed in Section 4, we determine the harmonic mean of soundness and completeness for each strategy as we increase the percentage of the feature space that is discarded. We reduce the feature space in increments of 5 percent until 95 percent of the feature space has been discarded.

5.3 Interpreting the Results

First we measured the performance of Automatch without any attempt at optimizing the dictionary through feature selection; that is, we use the Bayesian approach to score matches (Section 3.3) and the flow graph approach to optimize matches (Section 3.4). Using cross validation, we achieved a performance of 66% (measured as the harmonic mean of soundness and completeness). In a separate experiment, we used random guessing to match the same schemas and achieved a performance of 10%.

Next, we compared the three feature selection strategies of Section 4 and assessed their impact on schema matching. Figure 3 shows the performance for schema matching for each of the feature selection strategies. The x-axis is the percentage of low-scoring features that have been discarded, and the y-axis is the performance, measured as the harmonic mean of soundness and completeness. The leftmost point in the graph corresponds to our first experiment with no feature selection.

Initially, with 5% feature reduction, all the feature selection strategies improve performance by at least 6%. The strategies then perform comparably up to 60% reduction. At levels of reduction over 80%, IG and LR continue to produce improved matching performance (relative to no feature selection) while MI falls below performance with no feature selection.

All three feature selection strategies improve performance when compared to the initial performance with no feature selection (though the level to which they sustain this improvement varies). This observation indicates that *all* of these approaches are acceptable for reducing the feature space. Furthermore, if we are seeking the most ambitious reduction in the feature space, LR is preferable to IG which is preferable to MI.

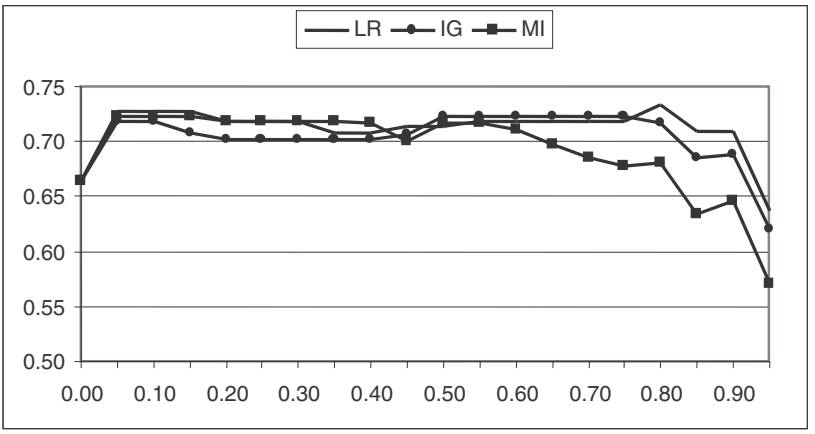

Fig. 3. Harmonic mean of soundness and completeness (y-axis) as the feature set is reduced in increments of 5 percent (x-axis)

6 Conclusion

In this paper we described an automated solution for the well-known problem of database schema matching. Our approach uses Bayesian machine learning, statistical feature selection, and the Minimum Cost Maximum Flow network algorithm to find an optimal matching of attributes between two semantically related schemas.

Our significant findings and contributions in this paper were:

- The Automatch system is a new and viable approach to eliminate the schema-matching bottleneck present in modern database applications. Our results are encouraging as they show performance that exceeds 70% (measured as the harmonic mean of the soundness and the completeness of the attribute matching process).
- Statistical feature selection can be used to improve the performance of Automatch. The improvement is in three areas: (1) in the *storage requirements* for the auxiliary knowledge base, (2) in the *computational costs* of the matching algorithm, and (3) in the *quality* (soundness and completeness) of the results. We estimate that statistical feature selection can be used to improve the performance of other automated schema-matching approaches (such as [6,10]) that must deal with high-dimensional feature spaces.
- Statistical feature selection incurs little overhead in Automatch since we are using a probabilistic learning approach. Learning after feature selection consists simply of normalizing the probabilities of the remaining features. In contrast, other machine learning approaches (e.g. neural networks, rule learners, etc.) must execute their respective learning algorithms after feature selection is completed.

While the performance of 70% in these experiments is promising, user interaction is still necessary to complete the matching process. In our future research, we plan on building a user interface that allows a domain expert to adjust the attribute mappings that have been proposed by Automatch. Furthermore, the interface will allow for iterative adjustment (i.e., after the user adjusts some of the mappings, we can re-apply Automatch for the remaining unmapped attributes).

An important benefit of user interaction in Automatch is that the system will be able to learn continuously. As new matches are provided through the user interface, the learner will be able to combine this information with what has already been learned. Note that this is significantly different than re-executing the entire learning algorithm. Such continuous learning is possible due to the statistical nature of the learning algorithm. As new matches are validated by a user, we can learn from these additional examples by updating the frequency counts of the features.

Finally, while this initial experimentation is encouraging, it is admittedly of a limited scale. Additional experimentation is planned to validate these preliminary conclusions.

Acknowledgement

The authors wish to thank Joseph (Seffi) Naor for his important suggestions in the area of network flows.

References

1. Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993. 458
2. Algorithmic Solutions. *The LEDA Users Manual (Version 4.2.1)*, 2001. 458
3. Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. ACM Press, 1999. 463
4. Jacob Berlin and Amihai Motro. Autoplex: Automated discovery of content for virtual databases. In *Proceedings of the Ninth International Conference on Cooperative Information Systems*, pages 108–122, 2001. 456
5. Silvana Castano and Valeria De Antonellis. A schema analysis and reconciliation tool environment for heterogeneous databases. In *Proceedings of the International Database Engineering and Applications Symposium*, pages 53–62, 1999. 453
6. AnHai Doan, Pedro Domingos, and Alon Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *Proceedings ACM Special Interest Group for the Management of Data (SIGMOD)*, 2001. 453, 464
7. Pedro Domingos and Michael Pazzani. Conditions for the optimality of the simple bayesian classifier. In *Proceedings of the 13th International Conference on Machine Learning*, pages 105–112, 1996. 456
8. Pat Langley, Wayne Iba, and Kevin Thompson. An analysis of bayesian classifiers. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 223–228, 1992. 456

9. Wen-Syan Li and Chris Clifton. Semantic integration in heterogeneous databases using neural networks. In *Proceedings of 20th International Conference on Very Large Data Bases*, pages 1–12, 1994. 454
10. Wen-Syan Li and Chris Clifton. Semint: A tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data & Knowledge Engineering*, 33(1):49–84, 2000. 454, 464
11. Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic schema matching with cupid. In *Proceedings of the 27th International Conferences on Very Large Databases*, pages 49–58, 2001. 453
12. Renée Miller, Laura Haas, and Mauricio Hernández. Schema mapping as query discovery. In *Proceedings of the 26th International Conferences on Very Large Databases*, pages 77–88, 2000. 452
13. Tom Mitchell. *Machine Learning*. McGraw-Hill, 1997. 456, 460
14. Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988. 461
15. Erhard Rahm and Philip Bernstein. On matching schemas automatically. Technical Report MSR-TR-2001-17, Microsoft, Redmond, WA, February 2001. 452, 453
16. Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A bayesian approach to filtering junk e-mail. *AAAI-98 Workshop on Learning for Text Categorization*, 1998. 460
17. Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 2000. 462