Mapping Maintenance for Data Integration Systems

Robert McCann, Bedoor AlShebli, Quoc Le, Hoa Nguyen, Long Vu, AnHai Doan

University of Illinois
{rlmccann, alshebli, quocle, hoanguyen, longvu, anhai}@cs.uiuc.edu

Abstract

To answer user queries, a data integration system employs a set of semantic mappings between the mediated schema and the schemas of data sources. In dynamic environments sources often undergo changes that invalidate the mappings. Hence, once the system is deployed, the administrator must monitor it over time, to detect and repair broken mappings. Today such continuous monitoring is extremely labor intensive, and poses a key bottleneck to the widespread deployment of data integration systems in practice.

We describe MAVERIC, an automatic solution to detecting broken mappings. At the heart of MAVERIC is a set of computationally inexpensive modules called *sensors*, which capture salient characteristics of data sources (e.g., value distributions, HTML layout properties). We describe how MAVERIC trains and deploys the sensors to detect broken mappings. Next we develop three novel improvements: *perturbation* (i.e., injecting artificial changes into the sources) and *multi-source training* to improve detection accuracy, and *filtering* to further reduce the number of false alarms. Experiments over 114 real-world sources in six domains demonstrate the effectiveness of our sensor-based approach over existing solutions, as well as the utility of our improvements.

1 Introduction

The rapid growth of distributed data has fueled significant interest in building data integration systems (e.g., [12, 22, 13, 15, 4, 2, 17]). Such a system provides users with a *uniform* query interface to a multitude of data sources, thereby freeing them from the tedious task of manually querying each individual source.

To answer user queries, the system uses a set of *semantic mappings* between the uniform interface (called *mediated schema*) and the local schemas of the data sources. Example mappings include "attribute cost of mediated schema matches price of a source schema", and "location matches address". The system uses the mappings in order to reformulate a user query into a set of queries on the data sources, then executes the queries and returns the combined results to the user.

Today, these mappings are created by the builders or administrators of the system, often in a laborious and error-prone process [25]. In dynamic environments, such as the Web, sources frequently change their query interfaces, data formats, or presentation styles [19, 21, 6]. Such changes often invalidate semantic mappings, causing system failure. Hence, once the system is deployed, the administrator must monitor it over time, to detect and repair broken mappings. Today such continuous monitoring is well-known to be extremely labor intensive [6, 28, 30]. In the long run, its cost often dominates the cost of system ownership [6]. Hence, developing techniques to reduce the maintenance cost is critical for the widespread deployment of data integration systems in practice.

In this paper we describe MAVERIC, an automatic mapping verification approach. MAVERIC probes a data integration system at regular intervals, and alerts the administrator to potentially broken mappings. In developing MAVERIC, we make the following innovations:

Sensor Ensemble: Given a data source S in the data integration system, we deploy multiple computationally inexpensive modules called *sensors*, each of which captures certain characteristics of S, such as distributions of attribute values, layouts of presentations, and integrity constraints over the data.

We probe source S while it is known to have the correct semantic mappings (to the mediated schema), then use the probe results to train the sensors. Subsequently, we apply the sensors to monitor the character-

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005

istics of S, and combine their predictions to verify the semantic mappings. We empirically show that our sensor ensemble approach significantly outperforms current mapping verification solutions.

Learning from Synthetic & External Data: Training over only the data of source S is often insufficient, because the sensors can only observe "normal" data. To make the ensemble of sensors more robust, we inject artificial changes into the source data, then use the perturbed data as additional training data for the sensors. We then extend the basic sensor framework so that in monitoring the mappings, it can also borrow training data from other sources in the data integration system, whenever appropriate.

Filtering False Alarms: In the final step, if the sensors report an alarm, we attempt to "sanity check" it, before reporting to the administrator. In essence, we take the data and compare it against known formats and values stored in the system, data at other sources, and data on the Web, to verify if it is still semantically correct. These filtering steps are computationally more expensive than the sensors, but are invoked only when an alarm is raised.

We empirically evaluated MAVERIC over 114 real-world sources in six domains. The results show that MAVERIC significantly outperforms existing approaches. It also demonstrates the utility of each individual MAVERIC component.

The rest of this paper is organized as follows. The next section defines the mapping verification problem considered in this paper. Section 3 discusses related work. Sections 4-7 describe the MAVERIC approach. Sections 8-9 present and analyze the experiments, and Section 10 concludes.

2 Semantic Mapping Maintenance

In this section we first discuss how data integration systems employ semantic mappings to answer user queries. We then discuss the need to maintain valid mappings, and define the mapping verification problem considered in the paper.

Mappings in Data Integration Systems: Figure 1.a shows a prototypical data integration system over three online real estate sources. Given a user query Q, the system translates it into queries over the source schemas, then executes them with the help of programs attached to the sources called *wrappers*. Figure 1.b illustrates this process in more detail, and highlights the role of semantic mappings and wrappers.

First, since query Q is posed over the mediated schema, a system module called *reformulator* [22, 17] consults the mappings between this schema and the schemas of data sources, to translate Q into queries at the sources. Suppose the query posed to a source S is Q_S (Figure 1.b).

In the second step, the wrapper associated with source S takes Q_S and executes it over the query interface of source S. In reply to the query, S produces a set of results in some presentation format, such as HTML pages (see Figure 1.b). The wrapper (not shown in the figure) converts these pages into a structured result set, say a relational table T_S. The reformulator then uses the semantic mappings again to convert T_S into a structured result set T_G in the vocabulary of the mediated schema. If T_G is the desired result for user query Q, then it is returned to the user. Otherwise, it is further processed (e.g., by joining with data at other sources [22]).

The Need to Maintain Valid Mappings: As described, it is clear that the semantic mappings play a crucial role in a data integration system. They are the "semantic glue" that enables query reformulation and data conversion. In dynamic environments, however, sources often undergo changes and invalidate such semantic mappings. Changes can happen with respect to:

- *Source availability or query interface:* A source can become unavailable or its query interface is redesigned. In this case the wrapper fails to query the source.
- *Source data:* A source may change the semantic meaning or representation of its data. For example, the meaning of price at a source changes from dollars to units of one thousand dollars. This will likely cause instances of price in the source results to be inconsistent with the meaning of cost in the mediated schema. Similarly, a source may choose to round all instances of price to whole dollars. Wrappers are notoriously brittle; a small change like this may prevent the wrapper from correctly identifying price instances.
- *Presentation format:* For example, a Web source may modify its template used to generate HTML result pages, by switching the order of attributes in each tuple, or changing the presentation of price instances from "$35,000" to "35,000 USD". Such changes often cause the wrapper to incorrectly extract query results.

Thus, when a source undergoes a change, the result produced by the wrapper (e.g., table T_S in Figure 1.b) often contains garbage data, or data whose semantic meaning has changed. This in turn makes the results returned to the user incorrect.

Despite the many ways invalid mappings can cause a data integration system to fail, there has been relatively little research on maintaining them (see Section 3). Currently integration systems are still maintained largely by hand, in an expensive and error-prone process. Consequently, a more efficient solution is needed to significantly reduce the cost of data integration systems.

The Mapping Verification Problem: Maintaining mappings requires two capabilities: detecting when a mapping becomes invalid, and repairing an invalid mapping. In this paper we address the former prob-

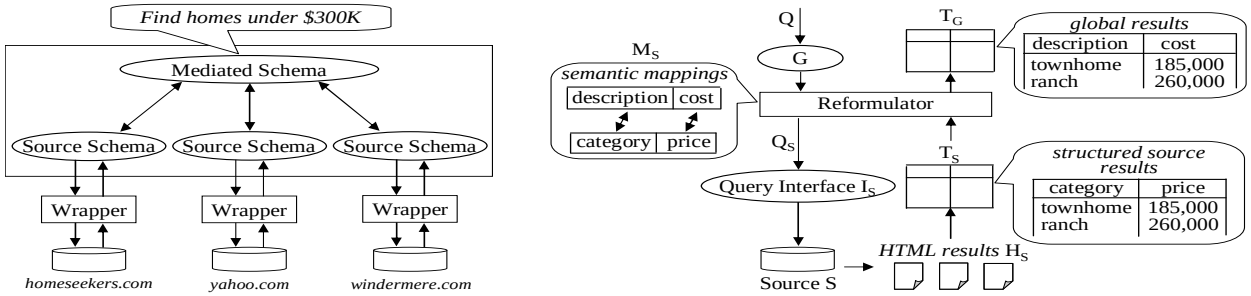

Figure 1: (a) A data integration system over three real estate sources, and (b) a closer look at querying a single data source.

lem, mapping verification. We assume that all mappings are initially valid (which is true when the administrator adds a source to the system for the first time). Our problem is then to monitor the data integration system and detect when any mapping has become invalid.

For ease of exposition, we will assume that sources export data in *HTML format*, which is converted by wrappers into *relational tables*. The solution we offer here however carries over to other presentation and structured data formats.

3 Related Work

Mapping and Wrapper Maintenance: The works most closely related to ours are [19, 21], which present solutions to wrapper verification. The work [19] leverages syntactic features, such as the average length of price instances. More recently, the work [21] leverages the same syntactic features as well as pattern features. For example, it learns that price has format "$XXX,000" in 95% of the training examples. The feature values and the number of matches for each pattern on newly probed data are compared to those on training data, and the wrapper is considered broken if the difference is significant.

Both of these works detect only syntactic changes, and are sensitive to small changes. For example, they may report broken mappings when sources change the syntactic representation of an attribute (*e.g.*, from "$185,000" to "$185K") while preserving its semantics. They also do not exploit HTML layout information, as well as possible integrity constraints between multiple attributes in a given tuple (*e.g.*, beds $\geq$ baths), as we do here.

Other works [24, 5, 31] focus on repairing a broken wrapper or mapping, and thus are complementary to Maveric.

Schema Matching: There is a large body of work on creating semantic mappings (*e.g.*, [25, 9, 7, 14, 16]). In [9] a learning ensemble is used, similar in spirit to the sensor ensemble used in Maveric. The idea is the same – to leverage multiple types of evidence for evaluating attribute semantics. However, the learning ensemble in [9] is performed once per source in an offline setting and the learning algorithms are typically too expensive for continuous verification of mappings across an entire data integration system.

Activity Monitoring: Mapping verification is also related to the broad topic of activity monitoring [11]. Well known problems in this area are fraud detection [3], intrusion detection [27, 20, 29], and detecting significant events such as recent news stories [1, 26]. The general problem is to monitor a data source, such as a stream of network packets or an online newspaper, and detect when a notable event occurs. Mapping maintenance can be interpreted as activity monitoring in which the source is a data integration system and notable events are invalid mappings. As such it is closest to the work [29], which detects an unauthorized computer user, and employs a sensor ensemble to model authorized users. This setting however is significantly different from ours. There, a continuous stream of feature values is monitored, such as CPU and network utilization, allowing the exploitation of continuous trends and reduction of false alarms by temporal filtering. For example, an alarm is sounded only if 65% of the last minute has been considered abnormal.

In our case, probing a data integration system is much more costly, hence each source is only probed periodically. Thus trends are not as useful and more sophisticated filtering schemes are needed to evaluate a distinct "batch" of data. We generalized the sensor ensemble to exploit a richer set of evidence (*e.g.*, page layout and source constraints), as well as developed a more expressive filtering scheme. Moreover, we developed two methods – perturbation and multi-source training – to overcome the scarcity of training data.

4 The Maveric Architecture

We now describe the Maveric architecture, which consists of four major modules: sensor ensemble, perturber, multi-source trainer, and filter. Maveric operates in two phases: training and verification. Figures 2.a-b describe the process of training and verifying a single data source S, respectively.

In training (Figure 2.a), Maveric starts by instantiating a sensor ensemble (which consists of a set of

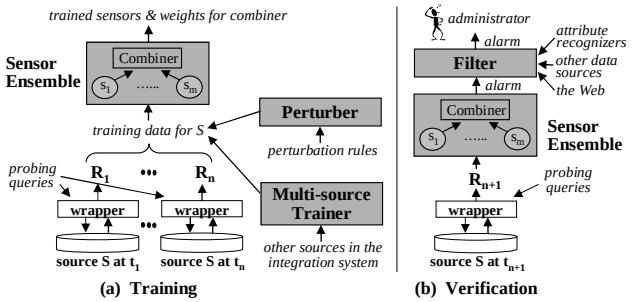

Figure 2: MAVERIC applied to a single source (a) during training and (b) during a verification time point.

sensors and a combiner) for source S. It then probes (i.e., queries) S over n time points in which it knows that the mappings of S are still correct. Next, it uses the query results to train the sensor ensemble. It also expands the training data using the perturber (to artificially change the probed query results) and the multi-source trainer (to obtain data from other sources).

Once training is done, MAVERIC enters the verification phase (Figure 2.b), in which it periodically verifies the correctness of the mappings of S. To do so, it probes S to obtain a set of query results, then feds the results to the sensor ensemble to compute a combined score. If this score exceeds a pre-specified alarm threshold, MAVERIC sends an alarm to the filter. The filter employs additional means (e.g., attribute recognizers, data from other sources, and the Web) to "sanity check" the alarm. If the alarm survives the checks, it is sent to the system administrator.

In what follow we describe the core architecture of MAVERIC: training and verification with the sensor ensemble. Sections 5- 7 describe perturbation, multi-source training, and filtering, respectively.

4.1 Sensor Ensemble: The Training Phase

Initialization: For each source S in the data integration system, MAVERIC begins by instantiating all applicable sensors, chosen from a set of sensor templates that we discuss in detail in Section 4.3. The result is an *untrained sensor ensemble* for S, consisting of all instantiated sensors and a generic combiner which combines the predictions of the individual sensors (see top of Figure 2.a).

Probe S to Generate Training Examples: Next, at each time point t_i ($i \in [1, n]$) during which MAVERIC knows that the mappings of S are still valid, it queries S with a set of queries $Q = \{q_1, \ldots, q_m\}$ to generate a training example R_i for the sensors.

Since we want the sensors to capture the characteristics of source S, we design Q to retrieve representative values for source attributes. For example, if S is a real-estate source, then for attribute price we may include three queries that retrieve houses priced under \$100K, between \$100K and \$200K, and over \$200K, respectively. While the three queries logically return all houses, we retrieve only the first few pages from the result of each query. For keyword attributes, such as house-description, we include queries that look for common words, such as "beautiful" and "view". We assume the set of probing queries Q is specified by the system administrator. In Section 8 we show that MAVERIC's accuracy is robust with respect to the choice of Q. The training example R_i then consists of

- all HTML pages retrieved by queries in Q, as well as the relational table returned by applying the wrapper of source S to these HTML pages. The sensors will examine both the HTML pages and the relational table to form "profiles" of S (see Section 4.3).
- the label "negative", meaning that the mappings are still valid. (In Section 5 we show how to create positive training examples with perturbation.)

The entire training set is then $\mathcal{R} = \{R_1, \ldots, R_n\}$.

Train the Sensors: Next, MAVERIC trains the sensors using the set of training examples $\mathcal{R}$. Intuitively, each sensor inspects $\mathcal{R}$ and builds an internal profile of valid query results for source S. The training process and thus the profiles are specific to each sensor type, and are discussed in Section 4.3. But the key to remember is that once trained, given any example R (which is the result of querying source S with queries in Q), a sensor s_i can inspect R and issue a confidence score on R being invalid (and thus the mappings of S being invalid).

Train the Sensor Combiner: Finally, MAVERIC trains the sensor combiner by computing for each sensor a *weight* that measures its verification ability. For this task, we use a variant of the Winnow algorithm [23]. This is a popular and fast weight-learning method [8] that has been successfully applied to related problems in combining predictions of "experts" of varying quality [8, 29]. Figure 3 describes the training algorithm. Briefly, it initializes the weight of each sensor to 1, then iterates. In each iteration it asks the resulting sensor ensemble to make a prediction for each training example in $\mathcal{R}$. If the ensemble prediction is incorrect, then it halves the weight of each sensor which also makes an incorrect prediction. In practice, Winnow is often run for a fixed number of iterations (e.g., to avoid overfitting) [8, 29].

4.2 Sensor Ensemble: The Verification Phase

MAVERIC verifies the mappings M_S of source S according to a pre-specified schedule (e.g., daily or weekly, as specified by the system administrator). We now describe the verification procedure at a single time point, t_{n+1}, as shown in Figure 2.b.

First, MAVERIC probes S with the same set of queries Q (described in Section 4.1 on training), to obtain a set of query results R_{n+1}.

Train the Sensor Combiner

Input: examples $R_1,...,R_n$ labeled with + or −, alarm threshold θ
sensors $s_1,...,s_m$ (already trained on $R_1,...,R_n$)
Output: sensor weights $w_1,...,w_n$
1. Initialize each weight w_i to 1
2. Repeat: for each example R_i
for each sensor s_j, $score_j \leftarrow$ the score of s_j when applied to R_i (Section 4.3)
$score_{comb} \leftarrow$ the combined score of all sensors using $w_1,...,w_n$ (Section 4.2)
if ($score_{comb} \geq \theta$ and $R_i.label = -$) // false alarm
$w_j \leftarrow w_j / 2$ for each $score_j \geq \theta$
else if ($score_{comb} < \theta$ and $R_i.label = +$) // missed alarm
$w_j \leftarrow w_j / 2$ for each $score_j < \theta$
until a stopping criterion is reached
3. Return $w_1,...,w_n$

Figure 3: Training the sensor combiner in MAVERIC.

Next, each sensor s_i examines R_{n+1} and produces a score $score_i$. The higher this value, the higher confidence s_i has that mapping M_S is invalid. Section 4.3 describes scoring functions for each sensor type.

Assume source S has m sensors, in the next step MAVERIC computes the weighted "vote" that M_S is invalid: $vote_{\text{invalid}} = \sum_{i=1}^{m} w_i \cdot score_i$, where the w_i are sensor weights learned with the Winnow algorithm (see Figure 3). It then computes the valid vote: $vote_{\text{valid}} = \sum_{i=1}^{m} w_i \cdot (1 - score_i)$. Finally, it computes the ensemble score as the normalized invalid vote:

$$score_{\text{comb}} = vote_{\text{invalid}} / (vote_{\text{invalid}} + vote_{\text{valid}}),$$

and outputs an alarm if $score_{\text{comb}} \geq \theta$, the alarm threshold used in Winnow (Figure 3).

4.3 The Sensors

MAVERIC uses the following types of sensors.

4.3.1 Value Sensors

These sensors monitor the value distributions of real-valued attribute characteristics. For each attribute A of source S, we instantiate seven sensors that respectively monitor (1) the number of instances of A in a result set, (2) the average number of characters per instance, (3) the average number of tokens per instance, (4) the average token length, (5) the percentage of numeric tokens, (6) the percentage of alphabetic tokens, and (7) the percentage of alphanumeric tokens.

Training: Let s be a value sensor that monitors feature (*i.e.*, characteristic) f of attribute A. Training s over a set of examples $\mathcal{R}$ means using $\mathcal{R}$ to build a profile of common values of f when A is valid.

In this work we use a Gaussian distribution for this profile. Specifically, we set the mean and variance of our profile to be equal to the sample mean and variance of f of A over the training examples. We chose the Gaussian family of distributions due to its success in related work ([19, 29]), even in cases where it was observed that this model did not fit the training examples very well. Investigating additional models is an important direction for future research.

Verification: During the verification phase, given a new set of query results R, recall from Section 4.2 that we must compute $score_s$, the confidence of sensor s that A is *invalid* in R (and thus the mappings have also become invalid). We can compute $score_s$ in two ways (in Section 8 we experimented with both):

- *Density Scoring:* $score_s = 1 - P(v)$, where v is the value of feature f of A in R and P is the density function of the Gaussian profile. Intuitively, the more frequent v is according to the profile of valid instances of A, the lower $score_s$ is.

This scoring method is simple to understand, but it fails to reflect the interpretation that a sensor score above 0.5 indicates that A is believed to be invalid in R' and a score below 0.5 signifies that A is valid (recall the real-valued voting scheme employed by the combiner in Section 4.2). For example, suppose that $P(v) = 0.2$. The above method yields $score_s = 0.8$, which is well above 0.5, indicating that A is invalid. However, it may be the case that v was the most common value of f in the training set and should be considered a strong indication that A is valid. Thus a better $score_s$ would be below 0.5.

- *Normalized Density Scoring:* To address this issue, we also investigate a normalized density scoring method. The idea is to compute a score based not solely on $P(v)$, but rather on how $P(v)$ compares to the densities for all other possible feature values.

Given v, we compute $score_s = Pr[P(v') \geq P(v)]$ where v' is also distributed by P. That is, $score_s$ is the probability that a random valid example will have a value v' for feature f with a higher density than v of the current example R. This method of scoring has the desired property that a "more common than average" feature value will output a low score (*i.e.*, indicate that A is valid), a strictly "average" value will output a score of 0.5 (*i.e.*, indicate complete uncertainty), and a "less common than average" value will output a high score (*i.e.*, indicate that A is invalid).

For the Gaussian profile, $score_s$ can be computed as $1 - 2 \cdot Pr[v' \leq \mu - |\mu - v|]$ using a look-up table for the cumulative Gaussian distribution.

4.3.2 Trend Sensors

These sensors monitor the trends in value fluctuation of attribute features. Specifically, for every value sensor s_v that monitors a feature f of attribute A, MAVERIC instantiates a trend sensor s_t that monitors $[f(R) - f(R')]$, the difference in the value of f between the current set of results R and the set of results R' obtained from the previous, last probing. The training and verification procedures for trend sensors are the same as those for value sensors (see Section 4.3.1), but with $f(R)$ being replaced by $[f(R) - f(R')]$.

4.3.3 Layout Sensors

These sensors monitor the *physical layout* of query results (e.g., in HTML format). For each source S that produces HTML pages, MAVERIC instantiates two layout sensors s_t and s_a, which monitor the *tuple ordering* and the *attribute ordering*, respectively.

To use these two sensors, during the training and verification phases, MAVERIC modifies the wrapper for source S so that when it extracts data fragments from an HTML page (to populate tuples in a relational table), it inserts wrapper tags into the HTML page to indicate the *locations* of those fragments. For example, the string "<WRAP:price_1>$185,000</WRAP:price_1>" in an HTML page indicates that "$185,000" is the value of price in the first tuple (of the relational table).

Monitor Tuple Ordering: The first layout sensor, s_t, requires no training. During verification, it examines the HTML pages marked up by the wrapper, as discussed above. If the markups show that distinct tuples overlap on the HTML pages (e.g., price for the second tuple is found between category and price for the first tuple), then the sensor outputs 1, indicating a broken mapping. Otherwise it outputs 0.

Intuitively, s_t exploits the tendency of tuples to be "horizontally separable" on HTML pages. If the newly probed HTML pages are not separable in this manner, the source presentation format might have been redesigned, causing the wrapper to incorrectly mark up data instances.

In the rarer cases where the correct HTML format is indeed not separable, sensor s_t will be "silenced" during the process of training the sensor combiner. If s_t causes the sensor ensemble to output false alarms, the training algorithm will detect the sensor's consistently high scores and exponentially quickly drive its weight toward zero. Thus s_t will have little effect on the output of the sensor ensemble during verification.

Monitor Attribute Ordering: The second layout sensor, s_a, monitors the attribute order within each extracted tuple. The sensor learns this order from the training tuples, again using the HTML pages marked up by the wrapper. Then during verification, it outputs 1 if the order has been changed, and 0 otherwise.

Intuitively, this sensor exploits the tendency of attributes to be consistently ordered in HTML pages. For example, Deep Web sources typically insert query results into an HTML page template where attribute order is fixed by a static set of "slots". However, for the same reason as discussed for s_t, this sensor will not harm the verification performance of the entire sensor ensemble for a source where the correct attribute order is not consistent.

4.3.4 Constraint Sensors

These sensors exploit domain integrity constraints that often are available over the mediated schema and source schemas. Example constraints include "house-area $\leq$ lot-area" and "price $\geq$ 10,000". Recent works have exploited such constraints for query optimization and schema matching purposes (e.g., [9]).

For each constraint C that is specified on the data source S, MAVERIC instantiates a constraint sensor. This sensor requires no training. During verification, it inspects the results of the probed queries, then outputs 1 if constraint C is violated, and 0 otherwise. C may be specified directly on the schema of S, or derived from one that is specified over the mediated schema (using the semantic mappings between the mediated and source schemas).

5 Perturbation to Improve Training

We have described how to train a sensor ensemble on a set of examples $\mathcal{R}$ retrieved from source S. This training process has two important shortcomings. First, the set $\mathcal{R}$ contains only "negative" examples, i.e., data where the mapping M_S is known to be correct. Having also "positive" examples, i.e., data where M_S is incorrect, can help make the sensor ensemble detect future positive cases more accurately.

Second, even the set of negative training examples in $\mathcal{R}$ is often not "sufficiently expressive". For instance, if until now a source has only displayed prices in the format "$185,000", then a trained sensor ensemble can only recognize this price format. If later the source changes its price format to "$185K", then the sensor ensemble will not recognize the new format, and will issue a false alarm.

To address these problems (which also arise in prior works [19, 21]), we propose to generate additional synthetic training data for the sensor ensemble, by perturbing the original training data $\mathcal{R}$ using common source changes. Example changes include reformatting instances of price from "$185,000" to "$185K", and switching the order of price and address in HTML pages. MAVERIC then trains the sensor ensemble on both the original and synthetic examples. We now describe this process in more detail.

5.1 Generate Synthetic Training Data

Recall from Section 2 that a data source can change its query interface, underlying data, or presentation format. In what follows we describe generating examples that model these changes.

Change the Query Interface: The query interface of S can become unavailable or redesigned. We approximate this change by assuming that the wrapper cannot submit queries to S, returning an empty result set. Thus, we form a single training example with empty data and a positive label, indicating that mapping M_S is invalid. (This is reasonable because if M_S is valid, a source will return empty result only if *all* probing queries return empty result, a very unlikely event.)

Change Source Data: S can change its data in two ways. First, it can add or remove tuples from the data. To model this change, for each original example $R \in \mathcal{R}$, we take its relational table T, then randomly remove and add some tuples to form new synthetic

examples R_1 and R_2, respectively. (Adding tuples to T is approximated by sampling tuples that already are in T.) Synthetic examples R_1 and R_2 receive the same label as that of R (which is "negative").

Second, S can change the underlying semantics of its data. For example, it may change the unit of price from dollars to thousands of dollars. We model this change as follows. Recall from Section 4.1 that each example $R \in \mathcal{R}$ consists of a set of HTML pages H and a relational table T (obtained by applying the wrapper to the HTML pages). We convert the HTML pages H into a new set of pages H', by changing all price values in H to reflect the new semantic meaning. For example, price value 23400 will be changed to 23.4, to reflect the unit change. Next, we apply the wrapper to H' to obtain a new relational table T'. H' and T' will form a new example R'.

Note that we cannot perturb table T directly to obtain table T'. When a source changes its data values or formats, we simply do not know how the wrapper will "behave" (i.e., what it will extract). Hence we must explicitly incorporate the wrapper behavior, by first simulating the changes on HTML pages H, thereby obtaining the set H', then simulating the wrapper behavior on H' to obtain T'.

Regardless of what the wrapper outputs for T', we already know that mapping M_S no longer holds, because the semantic meaning of price has been changed (from dollars to thousands of dollars). Hence example $\langle H', T' \rangle$ is assigned label "positive".

MAVERIC assigns to each attribute A_i of source S a set $\mathcal{C}_i$ of possible semantic changes (e.g., changing the unit meaning for price), then samples and carries out combinations of changes from the Cartesian product of the $\mathcal{C}_i$, in the above described fashion.

It is important to emphasize that the system administrator does not have to examine *all* attributes of all data sources to specify the sets $\mathcal{C}_i$. He or she specifies changes for only the attributes of the mediated schema. Then for each source S MAVERIC can employ the semantic mapping M_S to derive possible semantic changes for the attributes of S.

Change the Presentation Format: Source S can change its presentation formats in two ways. First, it can change the *layout*, for example, by switching the order of price and address. We model this change as follows. Again, let H and T be the HTML pages and relational table associated with a training example $R \in \mathcal{R}$, respectively. We switch the order of price and address in H, resulting in a new set H' of HTML pages. Next, we apply the wrapper of S to H' to obtain a new relational table T'. The new training example R' will consist of H' and T'.

In the next step, to obtain the label of R', we check if T' and T are equivalent (i.e., contain the same set of tuples). If so, then despite the order switch on HTML pages, the wrapper still works properly. So we assign label "negative" to R'. Otherwise, the order switch has broken the wrapper, resulting in label "positive" being assigned to R'.

Source S can also change the *format* of data instances in the HTML pages. For example, it can change prices: from "$185,000" to "185,000" or "$185K", or emails: from "abc@xyz" to "abc at xyz". We model such changes in a way similar to modeling layout changes, as described above.

5.2 Training with Perturbed Data

We now modify the sensor ensemble to train on both original and perturbed examples.

In Section 4.3 we presented four classes of sensors: value, trend, layout, and constraint sensors. To leverage the perturbed examples, we expand the training algorithms of the value and trend sensors. Layout and constraint sensors can be used without modification.

Let s be a value sensor which monitors feature f of attribute A. Without perturbation, s builds a Gaussian distribution P_- over the entire training set which contains only valid (i.e., negative) examples. With perturbation, this distribution is also built over all examples which are valid for A as well as a second distribution P_+ over all examples which are invalid for A (*i.e.*, generated via perturbation). Intuitively, these are profiles of valid and invalid instances of A, respectively. First, all training examples are partitioned into two subsets: $\mathcal{R}_-$ and $\mathcal{R}_+$, containing examples which are valid and invalid for A, respectively. Then P_- is built over $\mathcal{R}_-$ and P_+ is built over $\mathcal{R}_+$ as described in Section 4.3.

The modification for trend sensors is analogous, building two distributions over the change of f across consecutive time points. Here we compute the change of f only over consecutive examples generated by the same perturbation (and consecutive unperturbed examples).

The combiner is trained in the same manner as without perturbation (Section 4.1).

5.3 Verification with Perturbed Data

The last step is to modify the verification algorithms for value and trend sensors to leverage the two models built during training (Section 5.2). Let s be either a value or trend sensor monitoring feature f for attribute A. Given a new example R returned from the prober, the first step is to compute the value of f for A in R. Next, P_- is used to compute $score_-$ (Section 4.3), indicating the confidence of s that R is invalid (based upon the profile P_-). Using the same procedure, $score_+$ is computed using P_+. However, since P_+ is a profile of invalid instances, $score_+$ indicates the confidence of s that R is *valid* (based upon the profile P_+). Intuitively, $score_-$ and $score_+$ are quantifications of two sources of evidence (*i.e.*, two profiles) suggesting that R is invalid and valid, respectively. In

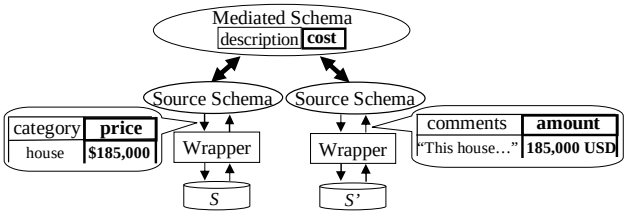

Figure 4: Borrowing data of attribute amount from source S' to help train for attribute price of source S.

this work we combine these into a single sensor score as $score = score_-/(score_- + score_+)$. That is, the sensor score of s is the confidence that A is invalid, normalized by the confidence that A is valid.

Note that this combination function lends the same credibility to $score_-$ and $score_+$. In practice, however, the number of training examples used to construct P_- and P_+ may differ significantly, suggesting that either $score_-$ or $score_+$ should play a larger role in determining the sensor score. One potential approach for further investigation is a scheme in which both $score_-$ and $score_+$ are weighted before applying the above combination function (*e.g.*, weighting $score_-$ by the number of training examples used to build P_-).

Note that if perturbation fails to generate invalid examples for A, there is no profile P_+. In these cases compute the sensor score as without perturbation (Section 4.3).

6 Multi-Source Training

For the sensors of a source S, we have discussed how to obtain training data directly from S (Section 4.1), or from perturbing S's data (Section 5). We now describe how additional training data can be "borrowed" from other sources in the data integration system.

Consider source S in Figure 4, whose schema contains attribute price. Recall that there may be multiple sensors that monitor price (Section 4.3). For example, a sensor may monitor the average number of characters in each price instance, while another one monitors the average length in token.

Since price (with format "$185,000") matches attribute cost of the mediated schema, which in turn matches amount of a source S', we can borrow instances of amount (which have the format "185,000 USD") to help train such sensors. After being trained on these new instances, the sensors will be better able to recognize valid instances of price if source S adopts the price format of source S' in the future.

To implement this idea, for each original training example $R \in \mathcal{R}$ which is associated with a relational table T, we replace all instances of certain attributes (e.g., price of table T) with instances of "equivalent" attributes from other sources (e.g., amount of S', see Figure 4). This results in a new table T'. Next, we check if all constraints of source S are satisfied on T'. If not, then T' is discarded, otherwise it becomes a new training example R'.

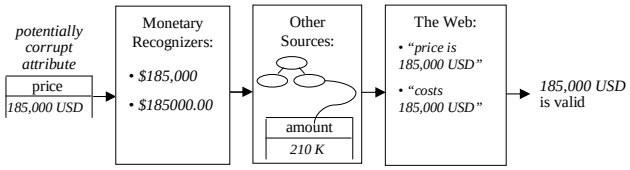

Figure 5: Filtering to remove false alarms.

Declaring Sensors Global or Local: In certain cases it may not be desirable to borrow training data from sources external to S. For example, consider attribute category of source S in Figure 4, which draws instances from a fixed vocabulary (e.g., house, commercial, lot, etc.). Suppose the system administrator wants MAVERIC to alert him/her whenever this meaning of category changes, then borrowing external training data for category may cause a problem. For instance, both category and the external attribute comments (of source S') map to description of the mediated schema (Figure 4). However, instances of comments are long textual paragraphs. Hence, borrowing these to train for category will likely mislead the sensors.

For this reason, MAVERIC allows the system administrator the option to declare each source attribute *local* or *global*. If a sensor involves any local attribute then it is declared *local*, otherwise *global*. MAVERIC trains local sensors only on the data probed and perturbed from their source, while for global sensors it also borrows training data from other sources.

7 Filtering False Alarms

The goal of MAVERIC is to detect when a mapping for source S has become invalid. Toward this end, we have described the sensor ensemble, a model of "normality" for the attributes of S. When the prober returns a new set of query results, MAVERIC sounds an alarm if the results do not fit this model well.

The challenge, however, is to define "well" such that invalid mappings trigger an alarm while valid mappings leave MAVERIC silent. The current standard solution (see the related work section) is to adjust a sensitivity threshold, such as the alarm threshold in MAVERIC. This solution however is inadequate. Setting the threshold too high risks not being able to detect many invalid mappings, which can have serious consequences to the operation of a data integration system. On the other hand, setting the threshold lower often generates a large number of false alarms, which drain the energy and resources of the system administrator. Indeed, false alarm has been a serious problem plaguing many verification systems [19, 21].

To reduce the number of false alarms, we propose filtering, a "sanity check" step to be added on top of current verification schemes. Figure 5 illustrates the process of filtering in MAVERIC. Suppose the sensor ensemble has output an alarm, saying in effect

that price has an unfamiliar format: "185,000 USD". MAVERIC then feeds the instances of price through a series of three filters, each of which attempts to check if the unfamiliar format can in fact be a valid format for price. (We will describe the working of these filters shortly.) If none of the filters recognizes the new price format, then the alarm is forwarded to the system administrator, otherwise it is silenced.

In what follows we describe three filtering methods currently employed in MAVERIC: recognizers, leveraging sources external to S, and utilizing the Web.

Employing Recognizers: When the sensor ensemble produces an alarm, MAVERIC inspects the individual sensor scores (see Section 4.3) to determine the set of attributes that have potentially been "corrupted". Then for each such attribute A, MAVERIC applies recognizers (if any) that have been designed specifically for that attribute type. A recognizer [9, 18] knows some common values or formats for A, and is often implemented as a dictionary or a set of regular expressions, for frequently occurring attributes such as person names, price, email, geographic location, color, date, etc. Figure 5 illustrates two monetary recognizers, which recognize the formats "$185,000" and "$185000.00".

A "corrupted" attribute is "silenced" if it is recognized by at least one recognizer. All "corrupted" attributes that have not been silenced are then forwarded to the multi-source filter.

Exploiting External Sources: This filter exploits data in sources external to S, in a way that is similar to multi-source training (Section 6) but differ from it in certain aspects. Specifically, suppose that sensor s raises an alarm on A and that A is a global attribute (see Section 6), then the filter attempts to leverage data in other sources to silence s. First, it retrieves instances of attributes (in other sources) that are equivalent to A. Note that it retrieves *fresh* data, rather than the existing training data at other sources. The reason is that since the last time the sensor ensemble was trained, sources might have changed and introduced new formats or values for instances of A. Thus the filter probes the sources to collect any possible "evidence" of these new formats and values.

The filter then creates a new sensor s' from the same sensor template as s, and trains s' on the collected data. Next, it applies s' to instances of A, and silences the original sensor s if the new sensor s' does not raise an alarm.

A "corrupted" attribute A is silenced if all sensors that raise alarm on it have been silenced. All remaining "corrupted" attributes are then forwarded to the Web-based filter.

Learning from the Web: Our final filter employs the Web to recognize unfamiliar instances of a "corrupted" attribute A, in a manner similar to that of the KnowItAll system [10], which collects instances of a concept (e.g., city, actor, etc.) from the Web.

Domain	Number of Sources	Schema Size (Number of Attributes)	Probing Schedule	Snapshots	
				Negative	Positive
Flights	19	8	weekly for 10 weeks	164	26
Books	21	6	weekly for 12 weeks	210	42
Researchers	60	4	daily for 313 days	12480	6274
Real Estate	5	17	11 snapshots per source	30	25
Inventory	4	7	11 snapshots per source	24	20
Courses	5	11	11 snapshots per source	30	25

Figure 6: Six real-world domains used in our experiments.

To explain this filter, consider again attribute price in Figure 5, with the unfamiliar instances such as "185,000 USD". In this example, the two monetary recognizers failed to recognize that these instances are prices. Exploiting equivalent attributes from other sources also did not help, because the format of amount is "210K", quite different from the format of price.

Thus, MAVERIC employs the Web to decide if "185,000 USD" is in fact a valid price instance. Toward this end, the Web-based filter first generates a set of *indicator phrases* such as "price is 185,000 USD" and "costs 185,000 USD". Note that some phrases are generic (*e.g.*, "<attribute> is <value>") while others are attribute specific (*e.g.*, "costs <value>" for monetary attributes). These templates are pre-specified in the filter. Next, the filter submits each of these indicator phrases to a search engine (*e.g.*, *google.com*) and records the number of hits returned. It also records the number of hits for the query consisting only of the instance value (*e.g.*, "185,000 USD").

Next, the filter computes a form of *pointwise mutual information* (PMI) between each indicator phrase and the instance-only phrase. For example,

$$\text{PMI}(\text{“costs <value>” and “185,000 USD”}) = \frac{|\text{hits for “costs 185,000 USD”}|}{|\text{hits for “185,000 USD”}|}.$$

Intuitively, a high PMI value suggests that "185,000 USD" is a valid instance of price. The filter averages the PMI scores of all available instances of price, to obtain a single score u.

Next, the filter computes a similar PMI score v, but over "junk" instances, i.e., those that are not valid instances of price. In our experiments we used instances of other attributes collected during training as "junk" instances for price. Then, if the quantity $u/(u+v)$ exceeds a pre-specified threshold, the filter considers the current instances of price to be valid, and silences the alarm over price.

By the end of the filtering step, if all "corrupted" attributes have been silenced, then MAVERIC silences the alarm raised by the sensor ensemble. Otherwise it sends the alarm to the system administrator.

8 Empirical Evaluation

We have evaluated MAVERIC on 114 sources over six real-world domains. Our goals were to compare

MAVERIC with previous approaches, to evaluate the usefulness of perturbation, multi-source training, and filtering, and to examine the sensitivity of MAVERIC with respect to its parameters.

Domains and Data Sources: Table 6 summarizes the six real-world domains in our experiments. "Flights" contains 19 airfare sites, "Books" 21 online bookstores, and "Researchers" 60 database researcher homepages (we will describe the other domains shortly).

For each data source S in these three domains, we first constructed a source schema, built a wrapper W, and provided a mapping M_S between the source schema and a mediated schema. We then periodically probed S with the same set of queries (taking care to ensure that the probing adapted to query interface changes at the source). This probing resulted in a set of snapshots $H_1, \ldots, H_n$, each of which is a set of HTML pages. Next, we labeled each snapshot H_i "negative" if applying the original wrapper W to it results in valid data (and thus valid M_S). Otherwise we labeled H_i as "positive". This gives us a set of labeled snapshots that we can use in evaluating MAVERIC. The last two columns of Table 6 shows the number of positive/negative snapshots in each domain.

The above three domains provide a "live" evaluation of MAVERIC, but this evaluation is limited to the changes that happened during the probing period. The next three domains ("Real Estate", "Inventory", and "Courses", see Table 6) enabled us to evaluate MAVERIC over a richer set of changes.

For each of these domains, we first obtained several large real-world data sets, which were previously retrieved from online sources (and now archived at *anhai.cs.uiuc.edu/archive*. We treated each data set as a source, and simulated the source being "live" by collecting an HTML page template from the Web, building an interface to return query results embedded in this template, and writing a wrapper to interact with this interface. This way we were able to probe the source as if it were currently online, but we could "evolve" the source as we like.

Next, for each source, we asked volunteers to provide five reformatted versions of the original HTML template, along with a correct new wrapper for each version. These HTML versions differed in several aspects: page layout, attribute format and display order, the use of auxiliary text, *etc.*. We then synthesized snapshots by pairing HTML templates with wrappers, and obtained for each source 11 snapshots: 6 negatives and 5 positives.

8.1 The Sensor Ensemble vs. Prior Solutions

We began by comparing the core architecture of MAVERIC, the sensor ensemble, with the "Lerman" system, a state-of-the-art wrapper verification approach [21] that was shown to outperform the system in [19].

Domain	Lerman System		Sensor Ensemble (D)		Sensor Ensemble (ND)	
	P / R	F-1	P / R	F-1	P / R	F-1
Flights	0.81 / 1.00	0.85	0.93 / 0.98	0.93	0.93 / 0.98	0.93
Books	0.83 / 1.00	0.89	0.90 / 0.99	0.93	0.90 / 0.99	0.93
Researchers	0.77 / 0.99	0.84	0.90 / 0.99	0.93	0.90 / 0.99	0.93
Real Estate	0.45 / 0.90	0.63	0.80 / 0.82	0.82	0.82 / 0.82	0.80
Inventory	0.52 / 0.89	0.67	0.75 / 0.90	0.77	0.71 / 0.90	0.75
Courses	0.49 / 0.94	0.66	0.92 / 0.88	0.88	0.88 / 0.87	0.85

Figure 7: Two variations of the core MAVERIC system compared to related work. Note that P, R, and F-1 are averages over multiple runs, and hence the formula F-1 = 2PR / (P + R) does not hold among these numbers.

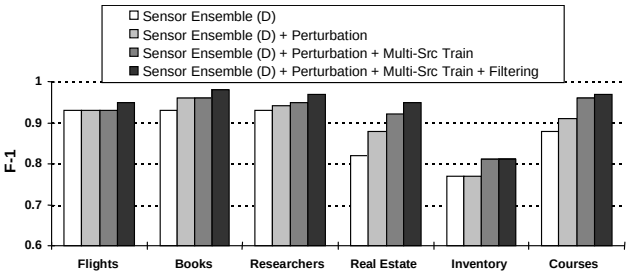

Figure 8: Accuracy of progressively enhanced versions of MAVERIC.

Figure 7 shows the accuracy of the Lerman system and two versions of sensor ensemble for MAVERIC: density scoring (D) and normalized density scoring (ND) as described in Section 4.3. Within each domain and for each verification system, we carried out three runs per source. In each run we trained the system over three negative snapshots (to simulate training over a period where the mapping is known to be correct), then applied it to verify the remaining snapshots (of that source). The accuracy of each run is measured with precision P = (number of alarms when snapshot is positive) / (number of alarms), R = (number of alarms when snapshot is positive) / (number of positive snapshots), and F-1 = 2PR / (P + R). To evaluate the potential of each system, we report results for the alarm threshold which maximizes F-1 performance (in Section 8.3 we show that MAVERIC is robust to varying this threshold). The reported P, R, F-1 are the averages across all runs in each domain.

The results show that MAVERIC significantly outperforms the Lerman system, increasing F-1 by 4-19% in each domain (Section 9 discusses reasons for improvements). Both the (D) and (ND) methods provide comparable results. Hence we use the (D) method for the remaining experiments.

8.2 Improvements to Maveric

In the next step we evaluated the utility of various enhancements. We started with the core MAVERIC, i.e., the sensor ensemble, then progressively added perturbation, multi-source training, and filtering. Figure 8 shows the accuracy for these four versions of MAVERIC (the experimental setup is identical to that of Section 8.1). The results show that each enhancement

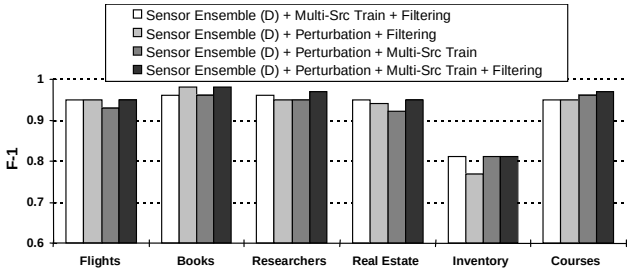

Figure 9: Evaluating the utility of each individual enhancement.

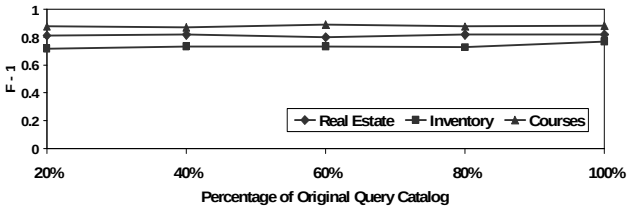

Figure 10: Accuracy of the sensor ensemble for varying numbers of probing queries.

improves the verification ability of MAVERIC. Adding perturbation improves over the sensor ensemble by up to 6% F-1, multi-source training additionally improves by up to 5%, and filtering additionally improves by up to 3%. Each enhancement improves F-1 in at least four of the six domains. The complete MAVERIC system (with all enhancements) reached accuracy of 82-98% F-1 across the domains.

To examine the utility of each individual enhancement to the complete MAVERIC system, we also measured the accuracy of four "stripped-down" versions of MAVERIC shown in Figure 9. The rightmost version is the complete MAVERIC. Each preceding version removes exactly one enhancement. The results show that each enhancement improved the performance in at least half of the domains, thereby contributing to the overall effectiveness of MAVERIC.

8.3 Sensitivity Analysis

Number of Probing Queries: Figure 10 shows F-1 accuracy of the sensor ensemble for varying numbers of probing queries. A data point at 60% means that we randomly sampled the original set of probing queries three times, each time taking out 60% of the queries, and using that as a new set of probing queries to rerun the sensor ensemble. The data point is the F-1 score averaged over the three runs.

The figure shows that F-1 varies by less than 2% in three domains and 5% in the remaining two (Inventory and Books), suggesting that the core MAVERIC is relatively robust for varying numbers of probing queries.

Number of Sensor Templates: In a similar experiment, we evaluated the sensor ensemble with only 80%, 60%, 40%, and 20% of the original set of sensor templates (used for experiments in Section 8.1). As the percentage goes to 20%, the results (not shown due to space limitation) show a steady decrease of F-1, by 10 or 19%, depending on the domain, suggesting that a rich set of sensors can significantly help MAVERIC improve accuracy.

Alarm Threshold & Duration of Training: In additional experiments we found that when varying the alarm threshold used in Section 8.1 by +/- 0.1, F-1 accuracy of the complete MAVERIC version varied by less than 0.12, and that F-1 changed gracefully with respect to the alarm threshold. This suggests that while the alarm threshold does affect accuracy, an administrator need not exactly optimize this threshold in order to achieve good verification performance with MAVERIC.

Finally, we have trained the sensor ensemble on up to ten snapshots, and found negligible change in F-1 accuracy. This is significant because training on fewer snapshots would place less burden over the system administrator.

9 Discussion

Reasons for Improvement: MAVERIC improves upon prior approaches due to several reasons. First, it exploits a broader collection of evidence. Prior works exploit only value and format features of data instances. In contrast, MAVERIC can exploit also layout and constraint related evidence. Second, MAVERIC has a highly modular design in the sensor ensemble, which enables the natural incorporation of these multiple types of evidence. Third, MAVERIC employs a combiner to explicitly evaluate the usefulness of each type of evidence, whereas previous works assume all evidence to be equally indicative of the validity of query results. For example, the average token length of (say) house-description is exploited by all verification approaches. However, prior works assume that the value of this feature is relatively stable across valid sets of query results. In practice, this value can fluctuate significantly over valid instances, in which case the combiner of MAVERIC will notice and place less emphasis on this feature during verification.

Beyond the core architecture, the three enhancements proposed in this work provide additional benefits. Perturbation and multi-source training generate additional instances beyond what is directly observable from a single source. This allows the sensor ensemble to build a broader notion of valid mappings and improve future predictions. Filtering also allows the use of more computationally expensive verification methods in order to reduce the number of false alarms sent to the administrator.

Limitations: In our experiments, MAVERIC failed to reach 100% accuracy for two main reasons. First, it encountered unrecognized formats. For example, in Courses it only learned (during training) that "2:00

PM" is a valid START-TIME format, so later it did not recognize that "1400" is also semantically equivalent. (Interestingly Web-based verification also did not catch this, because "1400" is used in many ways and hence does not have a high co-occurrence rate with TIME and START.) Other examples include "7/11/1996" vs. "July 11, 1996" and "M-W-F" vs. "Mon Wed Fri". Possible solutions include having a more exhaustive set of perturbation templates, and exploiting more sources, among others.

Second, MAVERIC encountered attributes with similar values. For example, a source in Inventory domain has ORDER-DATE and SHIP-DATE, both with format "7/4/2004". When the page was redesigned so that the order of these attributes was switched, the wrapper extracted them in the wrong order. But given that the extracted values had the same format, the current version of MAVERIC assumed this was correct. A constraint sensor such as one enforcing the constraint "ORDER-DATE $\leq$ SHIPPED-DATE" can help alleviate this problem.

10 Conclusions & Future Work

Monitoring semantic mappings to detect when they have become broken is a crucial task in deploying data integration systems. To this end, we have described the MAVERIC solution, which employs an ensemble of sensors to monitor data sources. We presented three novel improvements: perturbation and multi-source training to make the verification system more robust, and filtering to reduce the number of false alarms. Extensive real-world experiments demonstrated the effectiveness of our core approach over existing solutions, as well as the utility of our improvements. Besides further evaluation of MAVERIC, our main future work focuses on repairing the broken mappings once they have been detected.

Acknowledgments: This work was supported by the NSF grant CAREER IIS-0347903.

References

[1] J. Allan, R. Papka, and V. Lavrenko. On-line new event detection and tracking. In *R&D in IR '98*.

[2] R. Avnur and J. Hellerstein. Continuous query optimization. In *SIGMOD '00*.

[3] P. Chan and S. Stolfo. Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection. In *KDD '98*.

[4] J. Chen, D. DeWitt, F. Tian, and Y. Wang. Niagaracq: A scalable continuous query system for internet databases. In *SIGMOD '00*.

[5] B. Chidlovskii. Automatic repairing of web wrappers. In *WIDM '01*.

[6] W. Cohen. Some practical observations on integration of web information. In *WebDB '99*.

[7] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. imap: Discovering complex semantic matches between database schemas. In *SIGMOD '04*.

[8] T. Dietterich. Machine-learning research: Four current directions. *The AI Magazine*, 18(4):97–136, 1998.

[9] A. Doan, P. Domingos, and A. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *SIGMOD '01*.

[10] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A. Popescu, T. Shaked, S. Soderland, D. Weld, and A. Yates. Web-scale information extraction in knowitall. In *WWW '04*.

[11] T. Fawcett and F. Provost. Activity monitoring: Noticing interesting changes in behavior. In *KDD '99*.

[12] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. *Journal of Intelligent Inf. Systems*, 8(2), 1997.

[13] L. M. Haas, D. Kossmann, E. L. Wimmers, and J. Yang. Optimizing queries across diverse data sources. In *VLDB '97*.

[14] B. He and K. C.-C. Chang. Statistical schema matching across web query interfaces. In *SIGMOD '03*.

[15] Z. Ives, D. Florescu, M. Friedman, A. Levy, and D. Weld. An adaptive query execution system for data integration. In *SIGMOD '99*.

[16] J. Kang and J. Naughton. On schema matching with opaque column names and data values. In *SIGMOD '03*.

[17] C. Knoblock, S. Minton, J. Ambite, N. Ashish, P. Modi, I. Muslea, A. Philpot, and S. Tejada. Modeling web sources for information integration. In *AAAI '98*.

[18] N. Kushmerick. *Wrapper Induction for Information Extraction*. PhD thesis, 1997.

[19] N. Kushmerick. Wrapper verification. *World Wide Web Journal*, 3(2):79–94, 2000.

[20] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *SDM '03*.

[21] K. Lerman, S. Minton, and C. Knoblock. Wrapper maintenance: A machine learning approach. In *JAIR '03*.

[22] A. Y. Levy, A. Rajaraman, and J. Ordille. Querying heterogeneous information sources using source descriptions. In *VLDB '96*.

[23] N. Littlestone. Learning quickly when irrelevant attributes abound. In *Machine Learning 2*, 1987.

[24] X. Meng, D. Hu, and C. Li. Schema-guided wrapper maintenance for web-data extraction. In *WIDM '03*.

[25] E. Rahm and P. Bernstein. On matching schemas automatically. *VLDB Journal*, 10(4), 2001.

[26] D. F. Rahul. Monitoring the news: a tdt demonstration system.

[27] W. L. S. Stolfo, P. Chan, W. Fan, and E. Eskin. Data mining-based intrusion detectors: An overview of the columbia ids project. In *SIGMOD Record 30(4)*, 2001.

[28] L. Seligman, A. Rosenthal, P. Lehner, and A. Smith. Data integration: Where does the time go?, 2002.

[29] J. Shavlik and M. Shavlik. Selection, combination, and evaluation of effective software sensors for detecting abnormal computer usage. In *KDD '04*.

[30] L. D. Stein. Integrating biological databases. *Nature Rev. Genet.*, 4(5):337–345, 2003.

[31] Y. Velegrakis, R. Miller, and L. Popa. Mapping adaptation under evolving schemas. In *VLDB '03*.