WebIQ: Learning from the Web to Match Deep-Web Query Interfaces

Wensheng Wu, AnHai Doan
University of Illinois, Urbana, USA

Clement Yu
University of Illinois, Chicago, USA

Abstract

Integrating Deep Web sources requires highly accurate semantic matches between the attributes of the source query interfaces. These matches are usually established by comparing the similarities of the attributes' labels and instances. However, attributes on query interfaces often have no or very few data instances. The pervasive lack of instances seriously reduces the accuracy of current matching techniques. To address this problem, we describe WebIQ, *a solution that learns from both the Surface Web and the Deep Web to automatically discover instances for interface attributes.* WebIQ *extends question answering techniques commonly used in the AI community for this purpose. We describe how to incorporate* WebIQ *into current interface matching systems. Extensive experiments over five real-world domains show the utility of* WebIQ. *In particular, the results show that acquired instances help improve matching accuracy from 89.5% F-1 to 97.5%, at only a modest runtime overhead.*

1 Introduction

The World-Wide Web is often divided into the Surface Web and the Deep Web [11, 22, 12, 28]. The Surface Web consists of billions of browsable pages, while the Deep Web fields hundreds of thousands of data sources [6], such as *amazon*, *expedia*, and *realestate.com*.

Since Deep-Web data sources contain much valuable information hidden behind their *query interfaces*, many efforts have focused on querying and integrating the sources. Early works include [22, 8, 14, 16, 20] in the database and AI communities. Recent efforts include [11, 3, 12, 28, 26, 1, 18, 27], and recent industrial activities involve many startups, such as *Transformic*, *Glenbrook Networks*, and *Webscalers*. Given a domain of interest, such as book, movie, real estate, or air travel, an important focus of these efforts is to build a *uniform* query interface to the data sources in the domain, thereby making access to the individual sources transparent to users.

To build such a uniform query interface, a domain developer often must solve the *interface matching* problem: given a large set of sources in a domain, find semantic correspondences, called *matches*, between the attributes of the query interfaces of the sources [11, 28, 26]. Consider for example two query interfaces Q_a and Q_b in Figure 1. Example matches include attribute A_1 = From city of Q_a matching B_1 = Departure city of Q_b, A_5 = Airline matching B_3 = Carrier, and so on. Once the interfaces have been matched, approaches such as [27] can be employed to construct a uniform query interface and to facilitate querying the data sources.

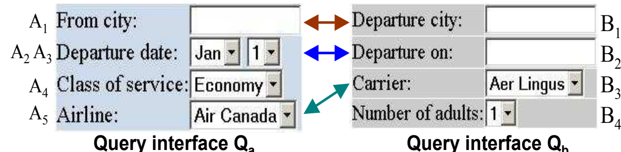

FIGURE 1: Two query interfaces in the air travel domain and semantic matches between them.

To match attributes of query interfaces, virtually all current solutions exploit the similarity between *labels* as well as that between *data instances*. Example labels include From city for attribute A_1 and Class of service for A_4 (see Figure 1). Example instances include Economy for attribute A_4 and Air Canada for A_5.

A major challenge facing these solutions, however, is the *pervasive lack of data instances*. Query interfaces often contain many attributes with no instance at all, such as attributes A_1, B_1, and B_2 in Figure 1. Indeed, for the five data sets used in our experiments (see Section 6), the percentage of attributes with no instance ranges from 28.1% to as high as 74.6%. For attributes that come with instances, the number of such instances is often small and even when the attributes match, their instances are often dissimilar. For example, the two attribute A_5 = Airline and B_3 = Carrier match, but the former lists instances that are mostly North American airliners (e.g., Air Canada) and the latter lists mostly European airliners (e.g., Aer Lingus).

Matching attributes with no or dissimilar instances is very challenging, since we can rely only on their labels, which are often generic or similar to many other labels. For example, the label of attribute B_1, Departure city, is similar to that of both A_1 (From city, a matching attribute) and A_2 (Departure date, a non-matching attribute). As another example, the two matching attributes A_5 = Airline, and B_3

= Carrier, have no common word in their labels.

It is important to note that the lack of data instances arises even in traditional schema matching contexts, such as during schema or view integration [23]. However, there the schemas to be matched often contain a variety of *other* meta-data information that can be exploited effectively by current matching techniques [23]. Examples of such meta data include attribute types, cardinality, the structural information among attributes, and semantic integrity constraints. In contrast, by their nature, query interfaces on the Deep Web contain very little or no such meta-data information. Hence, here the lack of data instances severely exacerbates the matching problem. Consequently, it is important to develop solutions that discover data instances for interface attributes, as these solutions can significantly improve the interface matching accuracy.

In this paper we describe WebIQ, a solution that learns from both the Surface Web and the Deep Web to automatically discover instances for interface attributes. The solution consists of the following three components:

Discover Instances from the Surface Web: Given an attribute A, such as Departure city, WebIQ formulates *extraction queries* such as "departure cities such as", using the attribute label and a set of lexico-syntactic rules [13]. For instance, a rule may specify that "if the label L is a singular noun phrase, then form the query '[plural form of L] such as"'.

Next, WebIQ poses the queries to a search engine (e.g., Google), to obtain a set of result snippets. Figure 2 shows such a snippet, in response to the above query.

FIGURE 2: A result snippet from Google.

WebIQ then examines the snippets to extract candidate instances. From the above snippet, WebIQ will extract three instances Boston, Chicago, and LAX.

Similar query-the-Surface-Web approaches have also been studied in the AI community, for example to populate ontologies [10]. However, in the context of interface matching, formulating extraction queries is significantly more challenging. This is because attribute labels often take syntactic forms that are not nouns or noun phrases, such as From city (a prepositional phrase). To address this problem, WebIQ performs a *shallow syntactic analysis* on the attribute label, using part-of-speech (POS) tagging [5] and pattern matching, then uses the analysis results to form appropriate queries. It also adds to such queries keywords formed from labels of other attributes, to narrow the scope of the queries.

Since the Web is often noisy, in the next step WebIQ must ensure that the extracted instances are indeed instances of the attribute. Toward this goal, it employs a two-phased validation process. First, in the *outlier detection* phase, WebIQ detects and removes false instances by performing discordancy tests [4], based on a set of type-specific test statistics. Second, in the *Web validation* phase, WebIQ forms a set of validation queries, using the attribute label, the extracted instance candidates, and a set of validation patterns. For example, a validation query for instance candidate Boston is "Departure city Boston". WebIQ then poses validation queries to the Surface Web, computes for each instance candidate a validation score, and returns those with sufficiently high scores. This two-phase validation process has an additional advantage that it greatly reduces the number of validation queries posed to search engines.

Borrow Instances from Other Attributes: Given an attribute A, WebIQ also attempts to "borrow" instances for A from other attributes. Specifically, suppose b is an instance of attribute B, then WebIQ will try to ascertain if b can also be an instance of A. Note that this can significantly help us to match A and B. In Figure 1, for example, WebIQ can try to ascertain if instance Jan of attribute A_2 = Departure date can also be an instance of attribute B_2 = Departure on, or if instance Aer Lingus of B_3 = Carrier can also be an instance of A_5 = Airline.

To verify that an instance b of attribute B is also an instance of A, one approach is to obtain a set of instances from the Surface Web for A, then check if b is among them. We found that this approach does not work well because b often is not in the top instances for A (as discovered from the Surface Web). Another approach is to form validation queries as described earlier, but using the label of A and the instance b, then check if the validation scores are comparable to those for the (existing) instances of A. We found that this approach does not work well because validation scores for b (e.g., Aer Lingus) are often much lower than those for the existing instances of A (e.g., Air Canada).

We observed that a more reliable way to assess the validation scores for b is to compare them with those for the *non-instances* of A (e.g., if A is Airline then Economy from attribute Class of service is a non-instance of A). The intuition is that the validation scores for the instances and the non-instances of an attribute are likely to be quite separable, and this *separation* can be exploited to accurately classify new instances. Based on this intuition, WebIQ first trains a validation-based classifier for A using the instances of other attributes on A's interface as negative examples, then employs the classifier to predict the membership of b.

Validate Borrowed Instances via the Deep Web: Consider again an attribute A. As discussed earlier, if A's label is not in "benign" syntactic form (e.g., noun or noun phrase), it may be difficult to formulate reliable extraction queries. Furthermore, the extraction queries may fail to ob-

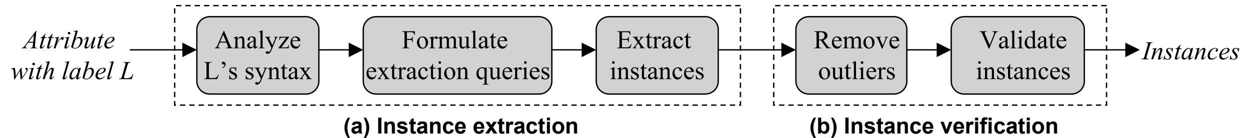

FIGURE 3: Steps in discovering instances from the Surface Web.

tain instances from the Surface Web.

In these cases, we can borrow instances for A from other attributes, as just discussed. However, it is unlikely that validating them via the Surface Web will work well, given that it is hard to formulate reliable extraction queries or that these queries have not returned instances.

To address this problem, WebIQ develops a solution to validate instances via the Deep-Web sources. Specifically, to verify that b is an instance of attribute A, WebIQ submits a query to the data source of A, with A's value set to b, then observes the response from the source. The key intuition is that in many cases the Deep-Web source will be able to distinguish instances of an attribute from non-instances even if the Surface Web cannot. For example, consider an attribute with label from (for the flight origin) on an airfare interface. While both from January and from Chicago might frequently occur on the Surface Web (thus making validating via the Surface Web difficult), often querying the source with attribute from set to Chicago will yield some meaningful results, whereas querying with from set to January will not.

In summary, we make the following contributions:

- A set of novel techniques, as embodied by the WebIQ system, that automatically acquire instances for attributes of query interfaces from the Surface Web and the Deep Web (Sections 2-4). The techniques also have potential applications in the general schema matching contexts (Section 8).

- The incorporation of the above techniques into a state-of-the-art interface matching system (Section 5).

- Extensive experiments over five real-world domains that demonstrate the utility of our techniques. In particular, the results show that acquired instances help improve matching accuracy from 89.5% F-1 to 97.5%, at only a modest runtime overhead (Section 6).

2 Discover Instances from the Surface Web

We now describe the three components of WebIQ. This section describes Surface, the component that discovers instances from the Surface Web, while the next two sections describe the remaining two components. Section 5 then discusses how the components are incorporated into IceQ [28], a recently developed interface matching system.

Given an attribute A and a constant k, Surface returns up to k instances of A, as gleaned from the Surface Web. It operates in two phases: extraction and verification (Figures 3.a-b, respectively).

In the extraction phase, Surface analyzes the syntax of A's label, and formulates a set of extraction queries. It then poses the queries to a search engine, obtains the results, and extracts instance candidates. In the verification phase, Surface first removes statistical outlier candidates, then verifies the rest of the candidates via the Surface Web. Finally, it returns the top k candidates, as ranked by their validation scores (if there are fewer than k candidates, then it returns all of them). The rest of this section describes the two phases in detail.

2.1 The Instance Extraction Phase

Analyze Label Syntax: As discussed earlier, an attribute label may take a variety of syntactic forms such as noun phrase (e.g., Departure city and Type of job), prepositional phrase (e.g., From and From city), verb phrase (e.g., Depart from), and even a sentence. Intuitively, it is relatively easier to formulate reliable extraction queries using nouns or noun phrases than other more open-ended forms such as prepositions. As such, this step analyzes an attribute label to extract nouns or noun phrases, which are then used in subsequent steps to form extraction queries.

Specifically, given an attribute A, Surface checks A's label for the occurrence of either a noun phrase, a prepositional phrase (a preposition followed by a noun phrase), or a noun phrase conjunction (a set of noun phrases connected by conjunctives such as "and" and "or", e.g., First name or last name). For a prepositional phrase, the noun phrase after the preposition is obtained. For a noun phrase conjunction, all noun phrases in the conjunction are obtained, and the rest of the instance discovery process is repeated for each noun phrase. If the label does not contain noun phrases, the extraction phase terminates and returns an empty set of instances.

To determine the syntactic form of the label, Surface employs a *shallow syntactic analysis* approach, which involves *part-of-speech (POS) tagging* and *pattern matching*. Specifically, first Brill's tagger [5] is employed to tag the label. The obtained POS tags are then matched against a set of pre-determined patterns to identify the interesting syntactic forms (as described above). For example, the pattern for noun phrases is:

> optional determiner + optional modifiers (adjectives/noun-adjectives) + noun + optional post-modifier (e.g., prepositional phrase).

Set extraction patterns:	
s1: *Ls such as* NP_1, ..., NP_n	s3: *Ls including* NP_1, ..., NP_n
s2: *such Ls as* NP_1, ..., NP_n	s4: NP_1, ..., NP_n, *and other Ls*
Singleton extraction patterns:	
g1: *the L of the O is* NP	g3: NP *is the L of the O*
g2: *the L is* NP	g4: NP *is the L*

FIGURE 4: Extraction patterns (L: label; Ls: L's plural form; NP: noun phrase; O: object name)

Such a pattern matching approach has been shown to be more accurate in many applications than more sophisticated syntactic parsing [17].

Formulate Extraction Queries: Given the noun phrases, this step formulates a set of extraction queries for attribute A. At a high level, we view instance discovery as a *question answering* problem, as commonly understood in AI: we pose a question, the extraction query, to a search engine to obtain a set of instances as the answer. The extraction queries can be regarded as incomplete sentences, and the job of the search engine is to complete the sentences with instances.

Specifically, Surface formulates extraction queries using the noun phrases obtained from the label of A and some domain information from the schema[1] of A. Domain information is used to narrow the scope of formulated queries as much as possible. We consider the following types of domain information:

- The name of the real-world entity that A is associated with (e.g., "book" on a bookstore interface).
- The name of the domain (e.g. "real estate" for a real estate interface), and
- The labels and instances of other attributes in the schema (e.g., "title" and "isbn" in a bookstore schema).

We note that the name of the object is typically the same as the name of the domain, and further that these information can be obtained automatically.

Extraction queries fall into two categories: *set* extraction queries and *singleton* extraction queries, with the former extracting a set of instances and the latter one instance at a time. The formulation of both types of queries is based on a set of *generic* extraction patterns listed in Figure 4, where s_i's (g_j's) are the *set* (*singleton*) extraction patterns, respectively. Note that the *set* extraction patterns are similar to those used in [13] for the acquisition of *hyponyms* from natural language texts. Each extraction pattern consists of two parts: *cue phrase* (shown in italic) and *completion* (NP or NP_i's). For example, the cue phrase in s_1 is Ls such as, where Ls is the plural form of the label L, and the completion is a list of noun phrases: NP_1, ..., NP_n, each considered to be an instance candidate for the attribute.

[1] In the rest of the paper, we use the terms "schema" and "query interface" interchangeably.

Given the set of extraction patterns, the extraction queries are formed using the cue phrases in the patterns. Specifically, for each pattern, its cue phrase is first *materialized* by replacing L with the noun phrase obtained from the label of attribute A. For example, suppose that A is an attribute in a bookstore schema and has a label author. Then, s_1 will generate authors such as and g_1 will yield the author of the book is. Next, the cue phrases are augmented with the domain information and properly formatted according to the query syntax of search engines, resulting in the final extraction queries. For example, one such extraction query to Google is

"authors such as" +book +title +isbn

where book is the name of the domain, title and isbn are the labels of some attributes in the schema. Note that double quotes enclose a phrase, while '+' signs request Google to ensure that the results contain the specified keywords.

Extract Instances from the Surface Web: The extraction queries are then posed to a search engine, which is Google in our experiments (using its Web API at www.google.com/apis). For each extraction query, we download top k snippets returned from Google. Finally, we employ a set of *extraction rules* to obtain instance candidates from the snippets. Each rule corresponds to one extraction pattern in Figure 4. An extraction rule consists of two parts: the first part identifies the *cue phrase* and the second part extracts the *completion*. For example, the extraction rule for the snippet in Figure 2 is: (1) identify the occurrence of the cue phrase "departure cities such as" in the snippet; and (2) extract the list of noun phrases which *immediately* follow the cue phrase, i.e., Boston, Chicago, and LAX.

2.2 The Instance Verification Phase

Remove Outliner Instance Candidates: Given a set of instance candidates, Surface prunes the set further in two steps: *pre-processing*, which determines the type of the instance domain and removes candidates which are not the determined type; and *type-specific detection*, which employs a set of type-specific *test statistics* to detect and remove further outlier candidates.

The pre-processing step employs a set of type-recognizing regular expressions to determine the type of the instance domain. Currently we consider only two types: numeric and string. If the majority of instance candidates (e.g., 80% in our experiment) are either monetary values (e.g., $15,200), integers, or real numbers, the instance domain will be determined to be numeric; otherwise it is string.

Next, the type-specific detection step performs *discordancy tests* [4] with a set of test statistics, all assumed to be

normally distributed. An instance candidate is considered to be an outlier if its test statistic is at least three standard deviations away from the average over all the candidates.

For instances of numeric type, the test statistics are their values. For example, it is unusual for the price of a book to be $10,000. For instances of string type, the test statistics are:

- the number of words in the instance, e.g., it is unusual for a person's name to have more than four words;
- the number of capital letters in the instance, e.g., the first letter of a city name is typically capitalized;
- the length of the instance (i.e., the number of characters in the instance), e.g., it is unusual for the make of a vehicle (such as Honda, Toyota) to have over 20 characters; and
- the percentage of numerical characters in the instance, e.g., the isbn of a book typically has ten digits and no more than three hyphens or white spaces.

Validate Instances via Surface Web: Web validation further removes false instances from the candidate set by assessing the *semantic* connection between the candidates and the attribute, based on their co-occurrence statistics on the Surface Web. The idea is that the meaning of an instance x can be partly characterized by the contexts where x appears. As such, if x is indeed an instance of attribute A, we expect that the label of A may frequently co-occur with x. Such *co-occurrence* statistics can then be exploited to measure the semantic connection between A and x.

As an example, suppose that A has label make (for automobiles). Consider Honda, one of A's instances. We would expect that make can often be found in the context of Honda in varied ways over the Surface Web pages, e.g., "a variety of makes such as Honda, Mitsubishi", "Make: Honda, Model: Accord", and "This car's make is Honda", as indicated by Google.

Based on the above observation, for each instance candidate x of attribute A, we form several *validation queries* using a set of validation patterns. Each validation pattern has two parts: a validation phrase V and the candidate x. Currently, we consider two types of validation patterns:

- *Proximity-based* pattern "L x", where $V = L$, the label of A. This pattern simply considers the proximity of L and x. For example, this pattern gives "make honda" as the validation query for L = make and x = Honda.
- *Cue phrase-based* patterns such as Ls such as x and such Ls as x, which utilize the cue phrases in the extraction patterns (Figure 4) as the validation phrases. For example, makes such as Honda is a validation query formed by these patterns.

Intuitively, validation phrases serve the purpose of distinguishing instances of an attribute from *non-instances*. In other words, we expect that instances of an attribute tend to occur more frequently with the validation phrases than non-instances. A possible measure on the co-occurrence of an instance with a validation phrase is the number of hits obtained from a search engine for the validation queries constructed as above. A problem with this measure is the potential bias towards popular instances (or non-instances).

To handle this problem, we adapt the *pointwise mutual information* (PMI) [10] to measure the co-occurrence. Specifically, consider a validation phrase V and an instance candidate x. Let $V + x$ be the validation query (which combines V and x). The PMI between V and x, denoted as $\mathrm{PMI}(V, x)$, is then given by:

$$\frac{\mathrm{NumHits}(V + x)}{\mathrm{NumHits}(V) * \mathrm{NumHits}(x)},$$

where $\mathrm{NumHits}(V)$ and $\mathrm{NumHits}(x)$ are respectively the number of hits obtained from a search engine on the validation phrase and the instance candidate, and $\mathrm{NumHits}(V + x)$ is the number of hits on the validation query. Intuitively, PMI between V and x measures the statistical dependence of V and x such that a larger PMI indicates a stronger dependence.

Denote the set of validation phrases for attribute A as $\mathcal{V} = \{V_1, V_2, ..., V_n\}$. The confidence score of x being an instance of A is then taken to be the average PMI score of x, i.e., $\sum_i(\mathrm{PMI}(V_i, x))/n$. Surface then returns the k instance candidates with top score.

3 Borrow Instances from Other Attributes

Given an attribute A, WebIQ can also "borrow" instances for A from other attributes. Specifically, suppose b is an instance of attribute B, then WebIQ will try to verify if b can also be an instance of A. This verification process can be done via the Surface Web or the Deep Web. This section describes Attr-Surface, the WebIQ component that verifies instances via the Surface Web. The next section describes Attr-Deep, the component that verifies instances via the Deep Web.

To verify if instances of B can be instances of A, Attr-Surface first learns an *instance classifier* for A from a training set, then employs the learned classifier to classify the instances of B. Many previous works on schema matching [9, 23] have utilized varied forms of instance classifiers, but they all rely on a large number of training examples. Such a large training set is not available from the interfaces, since an interface attribute typically has only a handful of instances available on its interface. Furthermore, it might be expensive to obtain a large number of instances from the Web. To address these challenges, we develop a novel approach to learning an instance classifier for an interface attribute. The learned classifier can be regarded as a variant of

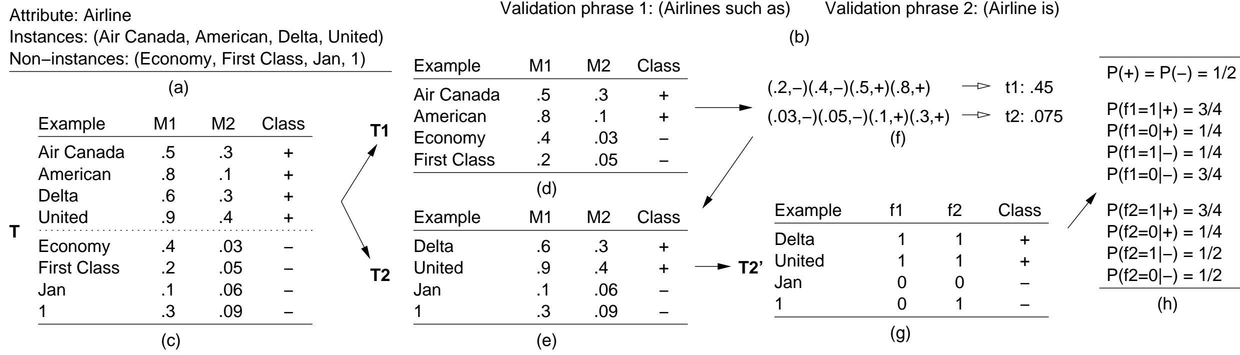

FIGURE 5: An example on training the validation-based classifier.

traditional naive Bayes classifier, but based on a validation scheme. Another distinct aspect of the approach is that the training of the classifier is fully *automatic*, with no needs for manually prepared training examples. We now describe the classifier and its training algorithm in detail.

3.1 A Validation-based Naive Bayes Classifier

A naive Bayes classifier [19] is a probabilistic function which, given an object represented by a feature-value vector and a finite set of classes, predicts the class membership of the object. The prediction is based on prior probabilities of the classes, class-conditional probabilities of the object, and the assumption that the features of an object are independent of each other given its class label.

More precisely, consider an object x represented by a vector $<f_1, f_2, ..., f_n>$, where f_i is the value of the i-th feature of x. Assume two classes: c and $\neg c$. The probability of x belonging to the class c, denoted by $P(c|x)$, is then given by:

$$\frac{P(c)\Pi_i P(f_i|c)}{P(c)\Pi_i P(f_i|c) + P(\neg c)\Pi_i P(f_i|\neg c)}. \qquad (1)$$

Clearly, features for a class should capture the *salient aspects* of the instances of the class so that they can be distinguished from the non-instances of the class. Our key observation is that the statistics obtained from the Surface Web on the validation queries for an attribute can be utilized as the features for the attribute. Specifically, we expect that the PMI scores of validation queries for instances of an attribute are likely to be much higher than those for non-instances of the attribute, and that this distinction can be exploited to perform classification.

Motivated by the above observation, we represent an object by its thresholded validation scores. Specifically, consider attribute A and an object x. Let $\mathcal{V} = \{V_1, ..., V_n\}$ be the set of validation phrases associated with A. First, we obtain x's validation scores and store them in a *validation vector* $M = <m_1, ..., m_n>$, where m_i is x's validation score on the i-th validation query of A. Let t_i be the threshold for the i-th validation score (which we show how to estimate in the next subsection). Then we use the t_i's to represent x with an n-dimensional vector $<f_1, ..., f_n>$, where $f_i = 1$ if $m_i > t_i$, and 0 otherwise. Intuitively, the thresholds characterize the separation in validation scores between the instances and the non-instances of A.

3.2 The Training Algorithm

Training the classifier for an attribute A amounts to estimating the probabilities in Formula 1. The training process can be divided into three steps: training set preparation, threshold estimation, and probability estimation. We now describe each step in detail. Figure 5 illustrates the process of training the classifier for the Airline attribute (A_5) on the interface Q_a shown in Figure 1.

1. Create Training Set T: First, we obtain a set of instances and non-instances for A. The non-instances of A are obtained from *other* attributes on the *same* interface as A. For example, Figure 5.a shows instances and non-instances of Airline used for the training. Next, for each instance of A, we obtain its validation scores as described in Section 2.2, using the Surface Web, and then turn it into a positive example. Similarly, we create negative examples using non-instances of A. For example, Figure 5.b shows two validation phrases associated with the attribute A_5, and Figure 5.c shows the obtained training set T, where the m_1 and m_2 columns show the first and second validation scores, respectively.

Finally, T is divided into two parts: T_1 and T_2, where T_1 is used to estimate the thresholds and T_2 to estimate the probabilities. For example, Figure 5.d and 5.e shows T_1 and T_2, respectively. Note that T_1 contains the first two positive examples and the first two negative examples in T.

2. Estimate the Thresholds: In this step, we use T_1 to estimate thresholds t_i's. Consider threshold t_i for the feature f_i. Intuitively, a good threshold should be the one that best separates positive and negative training examples in T_1. For this, we use *information gain* [19] to measure the quality of t_i. Specifically, suppose that t_i divides T_1 into T_{11} (where $f_i < t_i$) and T_{12} (where $f_i \geq t_i$). The information gain with respect to t_i is then computed as $E(T_1) - (|T_{11}|/|T_1| * E(T_{11}) + |T_{12}|/|T_1| * E(T_{12}))$, where $E(x)$ denotes the entropy of x. In other words, we choose

t_i such that it leads to the largest reduction in the entropy of the training examples in T_1. For example, Figure 5.f shows the derivation of t_1 and t_2.

3. Estimate the Probabilities: In this step, we first apply the learned thresholds t_i's on T_2 to transform each validation vector into a feature vector. This results in T_2' as shown in Figure 5.g. T_2' is then used to estimate the probabilities. Specifically, $P(f_i = 1|+)$ is estimated to be the percentage of positive examples in T_2' with $f_i = 1$. To avoid extreme 0/1 probability estimates, Laplacean smoothing [19] is applied. Other conditional probabilities are estimated similarly. Figure 5.h shows the estimated probabilities with the smoothing (e.g., $P(f_1 = 1|+) = (2 + 1)/(2 + 2) = 3/4$).

4 Validate Instances via the Deep Web

Besides validating instances via the Surface Web, as described in the previous section, WebIQ can also validate instances via the Deep Web. It implements this validation scheme in a third component called Attr-Deep. Given an attribute A and a borrowed instance x (of attribute B), Attr-Deep proceeds as follows.

Formulate and Submit a Query: First, a probing query is formulated by setting the value of A to x and the values of other attributes to their default values. Note that the query interface may contain some attributes that do not have instances. The default values for these attributes are typically empty strings. (Our experiments indicate that many interfaces permit partial queries where the values of some attributes can be left unspecified.) Next, the probing query is posed to the source.

Analyze the Response: This step applies several heuristics to analyze the response page from the source and determine if the submission was successful. We employ a variant of the heuristics used for a similar purpose in [22].

To reduce the number of queries to the source, if the submission is successful for at least one third of the instances of B, then we assume that all instances of B are instances of A.

5 Leverage WebIQ in a Matching System

We now describe how to incorporate the above three components of WebIQ into an interface matching system. The incorporation proceeds in two steps:

Instance Acquisition: Let $\{X_1, X_2, \ldots, X_n\}$ be the set of all attributes over all query interfaces. This step employs WebIQ to gather instances for these attributes. Consider attribute X_1. WebIQ gathers instances for X_1 as follows:

1. If X_1 has no instances, then gather instances for X_1 via the Surface Web, using the Surface component (Section 2).
   (a) If this gathering is successful, that is, at least k instances have been gathered, for a pre-defined k, then stop.
   (b) Otherwise, borrow instances for X_1 from $X_2, \ldots, X_n$ and validate them via the Deep Web, using the Attr-Deep component (Section 4). The reason that WebIQ does not validate them via the Surface Web is because it is unlikely to be successful, given that the instance gathering via the Surface Web in Step 1.a has been unsuccessful.
2. If X_1 has several pre-defined instances, then borrow instances for X_1 from $X_2, \ldots, X_n$ and validate them via the Surface Web, using the Attr-Surface component (Section 3). The instances cannot be validated via the Deep Web because X_1 accepts only *pre-defined* values. Thus, we cannot set the value of X_1 on the query interface to a borrowed value, if that value is not in the set of pre-defined values for X.

   Note that we can also obtain instances for X_1 via instance discovery on the Surface Web. However, we do not consider that possibility in the current scheme, to minimize the overhead caused by querying the search engine.

In Steps 1.b and 2, to minimize overhead, WebIQ does not borrow instances from *all* attributes. Instead, it borrows only from those attributes whose domains are deemed potentially similar to that of X_1. Specifically, consider an attribute X_i ($i \neq 1$) from a different interface as X_1. There are two cases: (1) X_1 does not have any pre-defined values (as in Step 1.b). In this case, the domain of X_i is likely to be similar to that of X_1 if the labels of X_1 and X_i are similar (so they are likely to match) and the domain of X_i is very different from the domain of any other attribute Y on the same interface as X_1 (intuitively, if Y and X_1 have similar domains, it is very unlikely that Y has some pre-defined values while X_1 does not). (2) X_1 has a set of pre-defined values (as in Step 2). In this case, the domain of X_i is likely to be similar to that of X_1 if there are at least two values, one from each domain, which are very similar.

The above steps are then repeated to gather instances for X_2, X_3, and so on.

Interface Matching: Once WebIQ has gathered instances for all attributes $X_1, X_2, \ldots, X_n$, an interface matching algorithm is employed as usual to match these attributes. In the current implementation, we use IceQ, a recently developed interface matching algorithm [28]. Briefly, IceQ employs interactive clustering to group attributes into clusters, each containing all attributes that match. To cluster, given any two attributes A and B, IceQ computes a similarity score based on the similarity of their labels and instances.

The similarity of their labels, denoted as $\text{LabelSim}(A, B)$, is given by $\text{Cos}(\vec{A}, \vec{B})$, where Cos is the *Cosine* function commonly employed in Information Retrieval and $\vec{X}$ denotes a vector of words

transformed from the label of attribute X. The similarity of their domains, denoted as $\mathrm{DomSim}(A, B)$, is evaluated based on the (inferred) types of the domains (such as integer, real, monetary values and date) and the values in the domains. Finally, the similarity of A and B, denoted by $\mathrm{Sim}(A, B)$, is computed as: $\mathrm{Sim}(A, B) = \alpha * \mathrm{LabelSim}(A, B) + \beta * \mathrm{DomSim}(A, B)$, where α and β are two constants (respectively set to .6 and .4 in our experiments, using numbers in [28]). During the clustering process IceQ can also interact with the user to automatically learn a thresholding value. However, in the current implementation we employ only the automatic version of IceQ, and set the threshold manually. See [28] for a detailed description of IceQ.

The above description of the similarity measure shows that its computation can benefit significantly from additional instances gathered by WebIQ. This benefit is also confirmed in our experiments in Section 6.

6 Empirical Evaluation

We have evaluated WebIQ using the ICQ data set in [28], which contains five real-world domains – airfare, automobile, book, job, and real estate, with 20 query interfaces in each domain. The first five columns of Table 1 show the characteristics of the data set. For each domain, it shows the average number of attributes per interface (Column 2), the percentage of interfaces containing attributes without instances (Column 3), and among these interfaces, the percentage of attributes without instances (Column 4).

Columns 3-4 clearly show that an overwhelming number of interfaces contain attributes with no instances (92% on average across the domains, as shown in the last row of Column 3). They further show that on average 40.7% of attributes in these interfaces have no instances. Thus, the lack of instances is pervasive.

We then manually examined that, for the attributes with no instances, whether it is reasonable to expect their instances to be found on the Surface Web, taking into consideration the fact that it is difficult to obtain instances for *generic* attributes (e.g., keyword and description) and attributes related to personal information (e.g., buyer id and reference number). Column 5 shows that on average, we can obtain instances for 89.6% of attributes, suggesting the potentials of an WebIQ-like approach.

Instance Acquisition: Next, for each domain we evaluated the effectiveness of WebIQ for instance acquisition, focusing in particular on attributes with no instance, since they are much harder to match than those with some pre-defined instances. For each such attribute, if WebIQ obtains at least 10 instances, then the acquisition process is deemed successful.

The last two columns (6-7) of Table 1 show the results. Column 6 shows the success rates when WebIQ employs only the component that discovers instances from the Surface Web (see Step 1.a in Section 5). Column 7 shows the success rates when WebIQ also employs instance borrowing and validation via the Deep Web (see Step 1.b in Section 5).

Domain	#Attr	IntNoInst (%)	AttrNoInst (%)	ExpInst (%)	Instance Extraction	
					Surface (%)	Surface+ Deep (%)
Airfare	10.7	85	32.2	100	19.0	81.1
Auto	5.1	95	28.1	100	58.7	82.2
Book	5.4	85	38.6	98	84.4	84.4
Job	4.6	100	74.6	83.1	72.2	72.2
Real Est	6.5	95	30.0	66.7	49.1	56.3
Average	6.7	92	40.7	89.6	56.7	75.2

TABLE 1: Characteristics of our data sets and results on gathering instances

Acquisition via the Surface Web: Column 6 shows the success rates between 19% in the airfare domain and 84.4% in the book domain, with an average of 56.7%. The airfare domain has a relatively low success rate because the labels of many attributes without instances are prepositions and verb phrases (e.g., from and depart from). As discussed earlier, it is very challenging to form reliable extraction queries for these attributes. Several attributes in the auto domain have very ambiguous labels (e.g., zip for "zip code"), reducing the success rate in that domain. Finally, the real estate domain has several attributes for measurement units (e.g., square feet and acreage), for which the extraction patterns are not as effective.

Both the book and job domains have very high success rates. This is not surprising, since the labels of most attributes with no instances in these domains are either nouns or noun phrases such as publisher, author, company name, and city. The extraction patterns tend to be very effective for these attributes.

Instance Validation via the Deep Web: Column 7 shows that this step significantly improves the success rates in both the airfare and auto domains. Interestingly, these are the difficult domains for getting instances from the Web. On the average, the success rate increases by 18.5%, demonstrating the effectiveness of validation via the Deep Web.

Interface Matching with WebIQ: In the next step we evaluated the extent to which WebIQ helps improve matching accuracy of IceQ (see Section 5). Following [28], we measure the matching accuracy via three metrics: precision, recall, and F-1 measure [25]. Precision P is the percentage of correct matches over all matches identified by the system, while recall R is the percentage of correct matches identified by the system over all matches given by domain experts. F-1 measure incorporates both precision and recall, and is computed as $2PR/(R + P)$.

For each domain, we performed three experiments. First, we collected the results of both IceQ and IceQ + WebIQ with no thresholding. That is, the clustering threshold of

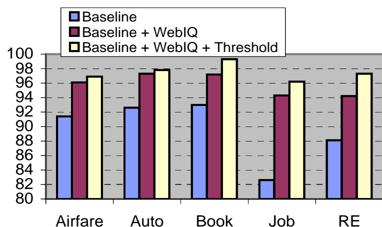

FIGURE 6: Matching accuracy

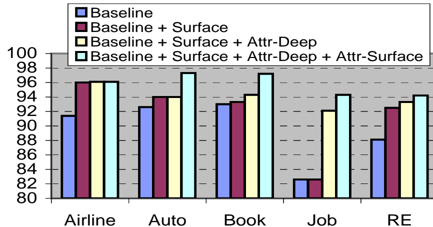

FIGURE 7: Component contributions

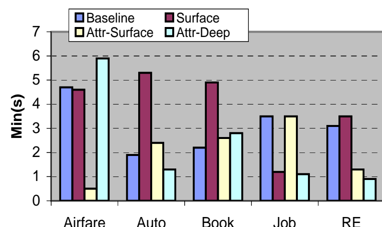

FIGURE 8: Overhead analysis

IceQ is set to zero so that as long as two attributes have a positive similarity, they may *potentially* be matched. This would allow us to directly compare our results with those in [28]. Then, the results of IceQ + WebIQ were recollected with the threshold τ uniformly set to .1 (which is about the average of the thresholds learned for the five domains in [28]).

Figure 6 shows the results. For each domain, it shows three bars which represent (from left to right) the F-1 accuracy of IceQ, IceQ + WebIQ, and IceQ + WebIQ with thresholding (in the figure, IceQ is referred to as "baseline").

The results show that IceQ + WebIQ significantly improved accuracy over IceQ, across all five domains. The improvement ranges from 4.2% in the book domain to 11.7% in the job domain. On the average, accuracy increases from 89.5% to 95.8%. The thresholding further increases accuracy to 97.5%. Detailed results indicate that most of the improvements with the thresholding were in the precision. This is not surprising given that the purpose of WebIQ is to increase the overall similarity of matching attributes by making their domains more similar.

Component Contributions: We further examined the contributions of the individual WebIQ components to the overall accuracy. Figure 7 shows the results, where the four bars for each domain represent the F-1 accuracy of baseline (i.e., IceQ), then baseline with the WebIQ components consecutively incorporated. Here, Surface refers to the WebIQ component that discovers instances on the Surface Web, Attr-Deep refers to borrowing and validating attributes via the Deep Web, and Attr-Surface refers to borrowing and validating attributes via the Surface Web.

The results show that Surface significantly improved matching accuracies (e.g., 4.6% increase in the airfare domain and 4.4% in the real estate domain). Attr-Deep had the most significant impact in the job domain (with a 9.5% improvement). Finally, Attr-Surface was very effective in four out of the five domains. On the average, it improved accuracy by 1.8%.

Overhead Analysis: Finally, we examined the overhead incurred by WebIQ. For each domain Figure 8 shows the times (in minutes) that IceQ + WebIQ spent matching attributes (the first bar), gathering instances from the Web (the second bar), validating instances via the Surface Web (the third bar), and validating instances via the Deep Web (the last bar). In other words, the last three bars show the overhead incurred by each individual component of WebIQ.

The results show that matching time ranges from 1.9 minutes in the auto domain to 4.7 minutes in the airfare domain. Time spent with Surface ranges from 1.2 minutes in the job domain to 5.3 minutes in the auto domain. This time varies in different domains due to different numbers of queries sent to Google. For example, the total number of extraction and validation queries for the job domain is 432 (over 20 source schemas). Note that the typical retrieval time from Google for one query is 0.1–0.5 second.

Time spent with Attr-Surface was at most 3.5 minutes (in the job domain), time spent with Attr-Deep was at most 5.9 minutes (in the airfare domain). The total overhead ranges from 5.7 minutes in the real estate domain to 11 minutes in the airfare domain. These results demonstrate that it is possible to employ WebIQ without incurring a significant overhead.

7 Related Work

Schema and data integration are important problems and have been extensively researched [23]. The problem of matching and integrating source query interfaces on the Deep Web has received much recent attention [12, 11, 28]. In [11], matches are identified by learning a generative model over a set of interfaces, but the model exploits only the statistics on the labels of the attributes. The importance of instances in matching interface attributes has been observed in both Wise-Integrator [12] and IceQ [28]. In particular, IceQ conducts a comparative study which shows that instances greatly improve matching accuracy.

Naive Bayes classifiers have been employed in many schema matching tasks [9, 23]. Compared to these conventional classifiers, a distinct aspect of validation-based naive Bayes classifier is that its features are based on the validation scores of the instances rather than the frequencies of words in the instances.

Question answering has been an active research area in both AI and IR communities (e.g., [15, 21, 24]). Our approach of gathering instances from the Surface Web is motivated in part by works on Web-based question answering such as AskMSR [2] and Mulder [15]. In particular, similar to Mulder and AskMSR, we also exploit the idea of

"redundancy-based extraction", where the scale and the redundancy of the information on the Surface Web are leveraged to extract answers to the questions from simple sentences, whose syntax is relatively easy to analyze.

There have been many works on information extraction [7, 10]. Many of them rely on the use of supervised learning techniques to train the system, while our approach of training instance classifiers for interface attributes is fully automatic.

Our approach of gathering instances from the Web is also inspired by the works on populating ontologies by exploiting the Web, such as KnowItAll [10]. But the task of gathering instances for interface attributes is more challenging as we have discussed. Furthermore, we believe that the techniques we developed for gathering instances of interface attributes such as label syntax analysis and outlier detection, can also be incorporated into other Web-based information extraction systems such as [7, 10].

8 Conclusion & Future Directions

We have described a set of novel techniques, as embodied by the WebIQ system, that automatically acquire instances for attributes of query interfaces from the Surface Web and the Deep Web. We showed how the techniques can be incorporated into an interface matching system. Extensive experiments over five real-world domains show the utility of our approach. In particular, the results show that acquired instances help improve matching accuracy from 89.5% F-1 to 97.5%, at only a modest runtime overhead.

Besides improving the effectiveness of the current solutions, our future work will study how to transfer our techniques to other contexts, such as mining the extensive bioinformatics literature to help match schemas of data sources in that domain, and mining text documents that accompany real-world database schemas for further metadata information. Overall, we believe the incorporation of shallow natural language processing techniques over corpora of domain data can greatly help semantic integration tasks, including matching Deep Web query interfaces, schema matching, and record linkage. Our current work is a first step in this direction.

Acknowledgment: We thank Google for facilitating our experiments. This work was supported by NSF grants CAREER IIS-0347903 and ITR 0428168.

References

[1] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 2003.

[2] M. Banko, E. Brill, S. Dumais, and J. Lin. AskMSR: Question answering using the worldwide Web. In *Proc. of 2002 AAAI Spring Symposium on Mining Answers from Texts and Knowledge Bases*, 2002.

[3] L. Barbosa and J. Freire. Searching for hidden-web databases. In *WebDB*, 2005.

[4] V. Barnett and T. Lewis. *Outliers in Statistical Data*. John Wiley & Sons, 1994.

[5] E. Brill. Some advances in rule-based part of speech tagging. In *AAAI*, 1994.

[6] K. C.-C. Chang, B. He, C. Li, M. Patel, and Z. Zhang. Structured databases on the web: Observations and implications. *SIGMOD Record*, 33(3), 2004.

[7] P. Cimiano, S. Handschuh, and S. Staab. Towards the self annotating web. In *Proc. of WWW*, 2004.

[8] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB*, 2001.

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

[10] O. Etzioni, M. Cafarella, et al. Web-scale information extraction in KnowItAll. In *WWW*, 2004.

[11] B. He and K. Chang. Statistical schema matching across Web query interfaces. In *Proc. of SIGMOD*, 2003.

[12] H. He, W. Meng, C. Yu, and Z. Wu. Wise-integrator: an automatic integrator of web search interfaces for e-commerce. In *VLDB*, 2003.

[13] M. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proc. of ICL*, 1992.

[14] N. Kushmerick, D. Weld, and R. Doorenbos. Wrapper Induction for Information Extraction. 1997.

[15] C. Kwok, O. Etzioni, and D. Weld. Scaling question answering to the Web. In *World Wide Web*, 2001.

[16] K. Lerman, S. Minton, and C. Knoblock. Wrapper maintenance: A machine learning approach. In *Journal of Artifi cial Intelligence Research*, 2003.

[17] C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.

[18] R. McCann, B. AlShebi, Q. Le, H. Nguyen, L. Vu, and A. Doan. Mapping maintenance for data integration systems. In *Proc. of VLDB*, 2005.

[19] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[20] M. Perkowitz, R. Doorenbos, O. Etzioni, and D. Weld. Learning to understand information on the Internet: An example-based approach. *J. Intelligent Information Systems*, 8(2):133–153, 1997.

[21] D. Radev, H. Qi, et al. Mining the Web for answers to natural language questions. In *CIKM*, 2001.

[22] S. Raghavan and H. Garcia-Molina. Crawling the hidden Web. In *Proc. of VLDB*, 2001.

[23] E. Rahm and P. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4), 2001.

[24] G. Ramakrishnan, S. Chakrabarti, et al. Is question answering an acquired skill? In *Proc. of WWW*, 2004.

[25] C. van Rijsbergen. *Information Retrieval*. Butterworths, London, 1979.

[26] J. Wang, J. Wen, F. Lochovsky, and W. Ma. Instance-based schema matching for web databases by domain-specifi c query probing. In *Proc. of VLDB*, 2004.

[27] W. Wu, A. Doan, and C. Yu. Merging interface schemas on the deep web via clustering aggregation. In *ICDM*, 2005.

[28] W. Wu, C. Yu, A. Doan, and W. Meng. An interactive clustering-based approach to integrating source query interfaces on the Deep Web. In *Proc. of SIGMOD*, 2004.