Automatically Labeling the Inputs and Outputs of Web Services

Kristina Lerman, Anon Plangprasopchok and Craig A. Knoblock
University of Southern California
Information Sciences Institute
4676 Admiralty Way
Marina del Rey, California 90292
{lerman,plangpra,knoblock}@isi.edu

Abstract

Information integration systems combine data from multiple heterogeneous Web services to answer complex user queries, provided a user has semantically modeled the service first. To model a service, the user has to specify semantic types of the input and output data it uses and its functionality. As large number of new services come online, it is impractical to require the user to come up with a semantic model of the service or rely on the service providers to conform to a standard. Instead, we would like to automatically learn the semantic model of a new service. This paper addresses one part of the problem: namely, automatically recognizing semantic types of the data used by Web services. We describe a metadata-based classification method for recognizing input data types using only the terms extracted from a Web Service Definition file. We then verify the classifier's predictions by invoking the service with some sample data of that type. Once we discover correct classification, we invoke the service to produce output data samples. We then use content-based classifiers to recognize semantic types of the output data. We provide performance results of both classification methods and validate our approach on several live Web services.

Introduction

Unprecedented growth in the amount of data available online has led researchers to develop novel information integration tools that can answer complex queries using data from online sources. In the bioinformatics domain, for example, these tools can help biologists seamlessly combine genomic, polymorphism and expression data with databases of phenotypic measurements to look for the genetic basis of diseases. In the geospatial domain, imagery, maps and vector data can be combined with other data to aid disaster management and recovery, among many possible applications. An information mediator framework — e.g., Prometheus (Thakkar, Ambite & Knoblock 2005) — provides uniform access to heterogeneous online sources, such as Web services, HTML pages or databases. A user must model the source by specifying its semantic definition: the type of data the source accepts and returns and the functionality the source provides. Once the source has been modeled, it can be used in an information integration plan to answer a specific query or composed with other sources to provide a new information service.

As new sources come online, we would like the information integration system to make full use of them without requiring the user to model them. While various technologies, most notably the semantic Web, have been proposed to enable programmatic access to new sources, they are slow to be adopted, and at best will offer only a partial solution, because information providers will not always agree on a common schema. Rather than rely on standards to which providers may or may not conform, we want to automatically model the functionality of a new service: i.e., learn the semantics of its inputs and outputs, and the operations it applies to the data.

Web services come with a syntactic description, contained in the Web Service Definition (WSDL) file. Unfortunately, the WSDL file normally does not contain enough information to allow the service to be used programmatically. We go beyond previous work in this area in that, given a new service (with a WSDL file), we automatically figure out how to invoke it, collect output data and learn the functionality of the service. Our approach is to formulate a hypothesis about functionality of a Web service and actively test the hypothesis by invoking the service and comparing the data it returns with that produced by services with known functionality. We divide the service modeling task into two problems: recognizing the semantic types of data used by the service (*semantic labeling*), and inducing the logical definition of the service (*learning functionality*) given the data types (Carman & Knoblock 2005). This paper addresses the first problem. Specifically, we describe machine learning techniques that leverage existing knowledge to learn the semantics of data used by Web services. While others have used metadata in the WSDL file to semantically classify the data types (Hess & Kushmerick 2003), we go further and verify a classifier's predictions by invoking the service. We generate examples for the input parameters using the predictions, and invoke the Web service with these data. Successful execution of the service both confirms the prediction and generates output data. We can then use novel content-based classification algorithms to semantically label the output data. In addition, we can combine the predictions of metadata and content-based classifiers to improve semantic labeling accuracy.

In the sections below we detail our solution to the prob-

lem of automatically labeling the input and output parameters used by Web services. This is an integral step towards creating a semantic model of a Web service. Since the methods for recognizing inputs and outputs exploit different sources of evidence — metadata in WSDL file and actual data returned by the service — we describe two classification methods that exploit different sources of evidence. The Section on "Automatically labeling input data" describes an algorithm that classifies input and output parameters based on terms in the Web Service Definition file. We call this metadata-based classification. The Section on "Automatically labeling output data" describes a novel content-based algorithm that classifies Web service's outputs based on their content. We present extensive evaluations of both approaches. Finally, we apply our methods to semantically label several live weather and geospatial Web services.

Related Work

The problem of identifying data types used by Web Services based on metadata is similar to problems in Named Entity Recognition, Information Extraction and Text Classification. Approaches that exploit surrounding information, such as adjacent tokens, local layout (Lerman *et al.* 2004a) and word distributions (Baker & McCallum 1998), have been used to assign a most likely class to the entity of interest. There are some aspects to the problem of classifying semantic data types of Web Services that make it distinct from those mentioned above. Usually, fewer tokens are used in naming a data type compared to those in documents. Even if one uses tokens from the corresponding Web Service messages and operations, the number is still small. We are, therefore, short on features useful for classifying data types. Secondly, texts in a WSDL file are generally ungrammatical, noisy and varied. Such situations cannot be tackled by previous solutions.

As information growth continues to yield abundant corpus samples, statistical machine learning approaches have been widely adopted. Various techniques have been developed to address the problems of data sparseness, inconsistency and noise. Several researchers have tried to categorize, or attach semantic information to, Web Service operations and data types. With respect to data type classification, rather than use metadata from WSDL files directly, (Hess & Kushmerick 2003) used those from HTML forms and metaphorically treated Web Form fields as the Web Service's parameters. Their assumption was based on a stochastic generative model. For a particular data type (or form field) D, Web Service developers use terms t drawn from a probability distribution $P(t|D)$ to name the semantic data type. Since it is infeasible to estimate parameters for $P(t|D)$ due to the terms' sparseness and huge vocabulary size, Hess and Kushmerick used the Naive Bayes' assumption to estimate $P(t|D)$: given a particular data type, terms are independent of each other. In our opinion, the trained classifier using this methodology is not accurate enough, probably due to the independence assumption and data sparseness. (Dong *et al.* 2004) proposed an unsupervised approach (Woogle) to cluster data types. The method is based on agglomerative clustering techniques which merge and split clusters using cohesion and correlation scores (distances), computed from the co-occurrence of terms within data types. This approach, however, has different a objective than ours: Woogle can roughly identify similarities or differences of data types; meanwhile, we need to still know the exact class in order to actively query a Web Service. In the Woogle approach, "Zip", "City" and "State" data types might be in the same group, Address, since they tend to occur together often. Meanwhile, we need to know exactly whether the parameter is "Zip" or "City" in order invoke the service correctly.

Our work is similar to schema matching or integration (Rahm & Bernstein 2001; Doan, Domingos, & Halevy 2003), where the objective is to produce a semantic mapping between instances of data from the two schemas. This is an important task in information integration, since queries expressed in some common language must be translated to the local schema of the database before they are submitted to the database. Past work on schema matching (Li & Clifton 2000; Doan, Domingos, & Halevy 2001; 2003) included machine learning techniques that learn to classify new object instances based on features that include local schema names and content features. The content features used in these works are global in nature, such as word frequencies and format. Our approach, on the other hand, uses finer-grained descriptions enabled by the use of patterns to describe the structure of data. Another distinction is that in schema matching, the data is usually available, while in our work, we retrieve the data by invoking services.

Semantic Labeling of Web Services

We use machine learning techniques to leverage existing knowledge in order to learn the semantics of new information sources. Our approach relies on background knowledge captured in the *domain model*. The domain model contains a hierarchy of semantic types, e.g., Temperature and Zipcode. These types correspond to objects in an ontology or their attributes. Our system knows how to populate the domain model with examples of each semantic type by querying the known definition of sources. For example, queries to $yahooweather$ return tuples of the form (Zipcode, Temperature, Wind, Humidity). The domain model is initially created manually. It is populated semi-automatically by users wrapping various data sources for their own applications (unrelated to the research problem described in this paper) and executing these Web wrappers. Our goal is to automatically relate the input and output types of new sources to the domain model.

Automatically Labeling Input Data

Web Services offer a natural domain for metadata-based classification techniques, because each service comes with a Web Service Definition file. The WSDL file specifies the protocols for invoking the service, lists supported operations, and specifies the data types of the inputs and outputs of the operations. Furthermore, an input may be constrained to a set of values, or facets, enumerated in the WSDL file. In most cases, however, operations, inputs and outputs carry no semantic information — only descriptive names attached to them by the Web service developer. We exploit these names

for classification. Our approach is based on the following heuristic: similar data types tend to be named by similar names and facets (if applicable), and/or belong to messages and operations that are similarly named (Hess & Kushmerick 2003; Dong *et al.* 2004). The classifier must be a soft classifier: it should not rigidly assign one class to a particular data type; instead, it should order all possible classes by likelihood scores, which will allow us to use examples from the next runner up class to invoke the service.

Hess and Kushmerick (2003) used a Naive Bayes classifier to assign semantic types to the input and output parameters. They represented inputs and outputs by terms $\vec{t}$ extracted from their names in the WSDL file. The classifier assigns an object represented by a feature vector $\vec{t}$ to class D (semantic type) that maximizes $P(D|\vec{t})$. $P(D|\vec{t})$ cannot be computed directly from data. Instead, we can estimate $P(\vec{t}|D)$, $P(\vec{t})$ and $P(D)$ from data, and then use them to compute $P(D|\vec{t})$ by using Bayes rule. Unfortunately, it is not feasible to compute $P(\vec{t}|D)$ (or $P(t_1, t_2, t_2, ..., t_n|D)$), because the number of possible combinations of terms will be exponentially large. To solve this problem an independence assumption is introduced: terms are assumed to be conditionally independent given a particular class D. Thus, estimation of $P(\vec{t}|D)$ is reduced to the estimation of $\Pi_{i=1}^{n} P(t_i|D)$. Laplace smoothing is applied in this setting to prevent zero-value estimation in the case that some terms do not co-occur with a certain class.

Potentially, because the independence assumption may not hold in this domain and Naive Bayes does not directly estimate decision boundaries, classification does not yield accurate enough results to enable the services to be successfully invoked. We thus ran another classification algorithm using Logistic Regression.[1] Without the term-class independence assumption, Logistic Regression directly estimates parameters, namely input weights, for computing $P(D|\vec{t})$ from the data (Ng & Jordan 2002; Mitchell 2005). The formula for computing $P(D|\vec{t})$ is $P(D|\vec{t}) = logreg(\vec{w}\vec{t})$ where $\vec{t}$ is a feature vector (occurrences of terms) and $\vec{w}$ is a weight vector. In the training phase, the classifier adjusts the weight vector to maximize log data likelihood, $\Sigma_{j=1}^{m} ln(P(D_j|\vec{t_i}; \vec{w}))$, where D_j is the class label of the j-th sample. One way to find such a weight vector is to use a gradient descent method. We use a logistic regression package written by T. P. Minka (Minka 2003) for this.

Eq. (1) gives the Logistic Regression equation for binary classification:

$$P(D = \pm 1|\vec{t}, \vec{w}) = \frac{1}{1 + \exp(-D\vec{w}^T\vec{t})} \qquad (1)$$

Here, $\vec{t}$ is a vector of features (terms), defined as $t_i = 1$ if feature t_i exists among the terms associated with the parameter we are classifying; otherwise $t_i = 0$. $D = 1$ if the parameter belongs to class D; otherwise $D = -1$, and $\vec{w}$ is a vector of weights to be learned. We used Eq. (1) to learn logistic classifiers for each input/output parameter.

Evaluation We evaluated metadata-based classification on a data set that consisted of 313 Web Service Definition files from web service portals (Bindingpoint and Webservicex) that aggregate services from several domains. We extracted names from several parts of the WSDL file — operations, messages, data types and facets. Subsequently, the names were decomposed into individual terms. Thus, GetWeatherByZipRequest was decomposed into five terms — get, by, request, weather, zip — where the first three words are stopwords and are ignored. Each extracted term was then stemmed with Porter Stemmer. Each input and output parameter was represented by a set of features: terms from its operation, message name and facets. We expanded the complex data types to their leaf nodes and used those as data types; meanwhile, terms from objects at higher levels were collected as "auxiliary features." In all, $12,493$ data types were extracted and labeled. Each was assigned to one of 80 classes in the geospatial and weather domains: "latitude," "city," "humidity," etc. Other classes, e.g. "passenger," were treated as "unknown" classes.

Both Naive Bayes and Logistic Regression classifiers were tested using 10-fold cross validation. Both classifiers were trained using a one-against-all method, and predicted classes for each data type were ordered by their probabilities. Since data types from the same message generally tend to contain similar terms, to avoid introducing classification bias, data types from the same message were forced to be in the same fold. Alternatively, we ensure that all data types from a WSDL file are in the same fold, as it is likely the service developer used similar terminology for different data and operation names. Table 1 presents classification accuracy results for the two classifiers. Results are shown for increasing tolerance levels. Zero tolerance ($T = 0$) means that highest ranked prediction was the correct one; $T = 3$ means that correct class was among the top four guesses. The logistic regression classifier significantly outperformed Naive Bayes.

Automatically Labeling Output Data

We developed a domain-independent pattern language (Lerman, Minton, & Knoblock 2003) that allows us to represent the structure of data as a sequence of tokens or token types and learn it from examples. These can be specific tokens,

Table 1: Classification accuracy for the Naive Bayes and Logistic Regression classifiers

	$T = 0$	$T = 1$	$T = 2$	$T = 3$
	(a) fold by wsdl			
Naive Bayes	0.64	0.80	0.84	0.86
Logistic Regr	0.87	0.94	0.95	0.96
	(b) fold by message			
Naive Bayes	0.65	0.84	0.88	0.90
Logistic Regr	0.93	0.98	0.99	0.99

[1] Logistic Regression and Naive Bayes classifiers are related: One can prove that Naive bayes can be represented in Logistic Regression form (Mitchell 2005). Also, if the independence assumption holds and there are infinite training data, both classifiers will give the same result (Ng & Jordan 2002).

such as "90210," as well as general token types. The general types have regular expression-like recognizers, which simply identify the syntactic category to which the token's characters belong: 5DIGIT, NUMBER, CAPS, etc. The symbolic representation of data content by patterns of tokens and token types was shown to be concise and powerful. These patterns can be efficiently learned from examples of a data field. We found that learned patterns enabled a system to locate examples of the field on Web pages with considerable accuracy in a wide variety of information domains (book sellers, phone books, ...) (Lerman *et al.* 2004b; Lerman, Minton, & Knoblock 2003).

In a nutshell, the pattern learning algorithm finds all statistically significant patterns that describe the data. The learning algorithm is biased towards producing more specific patterns. For example, suppose we are trying to learn the description of Zipcode. If the specific pattern "90210" and its generalization "5DIGIT" are both significant, the algorithm will keep the more specific pattern in addition to the general pattern. This tends to produce content descriptions comprised of many patterns. In addition to patterns, we describe the content of data by two other features — the mean and variance of the number of tokens in the training examples.

We can use learned patterns to recognize semantic types of Web service's output parameters based on the content of data it returns. The basic premise is to check how well the set of patterns associated with each semantic type (from the domain model) describes new examples, and assign them to the best-scoring semantic type. We developed heuristics to score the match. Factors that increase the score are:

- Number of patterns that match examples[2]
- Pattern weight — higher for more specific patterns
- Pattern length — penalty for unmatched tokens

The output of the algorithm is top four guesses for the semantic type, sorted by score.

Evaluation We validated content-based classification on data sources from a variety of domains. We used existing wrappers created by different users to extract attributes from these sources. We reserved two or three sources from each domain for learning data patterns, and then used content-based classification to semantically label data from the remaining sources. The sources and extracted attributes were:

Airlines domain: Five sources related to flight status, with 44 attributes: airport, airline, time, date, flightstatus, ...

Directory domain: Five directory services with 19 attributes extracted: fullname, streetaddress, city, state, zipcode, and phonenumber

Geospatial domain: Eight sources, 21 test attributes extracted: latitude, distance, streetaddress, zipcode, ...

Electronics domain: Twelve sources related to electronics equipment. 137 attributes extracted, such as modelname, manufacturer, modelnumber, hsize, vsize, brightness, power, colordepth, scanresolution, weight ...

[2]To be considered a match, a pattern has to match a prefix of an example, not all the tokens in the example

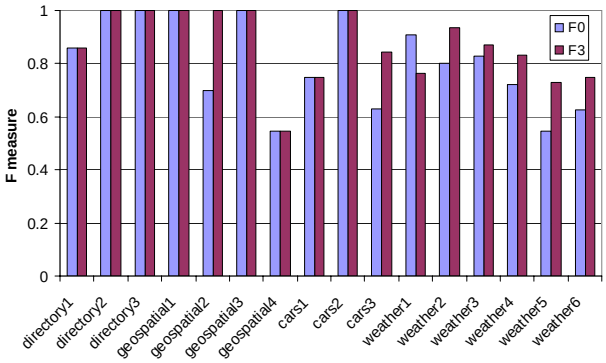

Figure 1: Performance of the content-based classification algorithm on data from a variety of domains.

Cars domain: Five sources with 23 test attributes extracted: make, model, year, color, engine, price, mileage, bodystyle. Because most of the data was alphabetic, and each site had its own capitalization convention, we converted all data to lower case strings.

Weather domain: Ten sources with 87 test attributes: temperatureF, tempInF, sky, wind, pressure, humidity, ...

Figure 1 shows performance of the content-based classification algorithm on data extracted from a variety of sources. The F-measure is a popular evaluation metric that combines recall and precision. Precision measures the fraction of the labeled data types that were correctly labeled, while recall measures the fraction of all data types that were correctly labeled. F0 refers to the case where the top-scored prediction of the classifier was correct, and in results marked F3, the correct label was among the four top-scoring predictions. We believe that even if the algorithm cannot guess the semantic data type correctly, if it presents the user with a handful of choices, which contain the correct type, this will speed up the source modeling task.

Content-based classification attains very good performance on the data sources shown in Figure 1, scoring at least $F = 0.80$ on 12 of 16 sources. We also applied the algorithm to label data from sources in two other domains, airlines and electronics equipment, but with poorer results, as shown in Figure 2(a). The electronics equipment domain was very hard. Most of the fields in the training data contained only numeric tokens (e.g., "1024" for horizontal display size). The learned patterns did not contain units information; therefore, they could not discriminate well between fields. In the airlines domain, precision suffered for the following reasons: non-standard capitalization of text fields, such as flightstatus, and incompatible date formats in training and test data. One way to improve precision and recall is by considering semantic types from a single domain only. The domain of a Web service can be identified in a number of ways, most directly from keywords in the WSDL file, or by analyzing semantic types assigned to data fields by the classifier and choosing the domain to which most of the semantic types belong. Figure 2(b) shows results of restricting

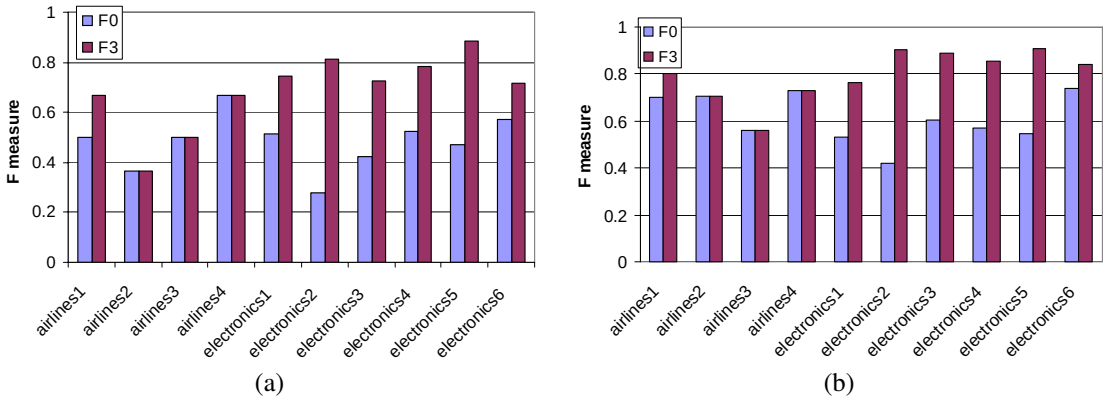

Figure 2: Performance of content-based classifier on data from electronics and airlines domains when (a) considering all semantic types in classification, (b) when restricting semantic types to the domain of the source

the content-based classification algorithm to semantic types within one domain only (e.g., electronics equipment). As expected, performance of the algorithm improves.

Empirical Validation

To evaluate our approach on live Web services, we searched for Web services in the Weather and Geospatial domains. Since quite a few services restrict access to registered users, we only tested those services for which we obtained license keys. Furthermore, we filtered out services that only provide information about places outside of the United States. This left eight invocable services. Several of the services supported more than one operation; therefore, we attempted to semantically model data types used by each of the 13 operations.[3]

First, we classify the input parameters used by the services. Then, we can generate sample values for the inputs and invoke the service with them. Our only information about the service is the WSDL file; therefore, we employ metadata-based classification to semantically label input parameters. We retrieved each service's WSDL file and processed it to extract features associated with each input and output. Next, we classified them using the Logistic Regression classifier (trained on all WSDL files from Bindingpoint and Webservicex directories). We then generated sample data for the input parameters. We took classifier's top guess as the semantic type of input parameter and chose a random sample of this type from data in the domain model.[4] We invoked the service with sample input data. If classifier's top guess was "Unknown" (the semantic type outside of our domain model), we left the corresponding input parameter blank. If we could not invoke the Web service successfully with the top guesses for the input parameters, we went back to classifier results and chose the next best guess. Clearly, if there are many input parameters, this combinatorial search will be expensive to execute. Fortunately, in the chosen domains, each Web service had one or two mandatory input parameters.

We classified 47 input parameters. The classifier chose "Unknown" as the most likely class for four inputs from two services. For one of them (Arcweb's AddressFinder), the "Unknown" parameters were optional (intersection, zipcode), and the service was successfully invoked. The other service (Terraworld) could not be invoked because a mandatory input parameter (streetaddress) was misclassified.

We successfully invoked seven of the eight services (or 12 of 13 operations), collected 3–6 samples for each output parameter and classified them using content-based classification algorithm. The classifier labeled 168 out of 213 output parameters. The classifier's top prediction was correct in 107 cases, leading to accuracy $A = 0.50$. These results are significantly worse than ones quoted in the previous section. This is due to an order of magnitude difference in sample sizes. We believe that performance will improve once we automate the data collection process and obtain more data samples from the services.

Sometimes the metadata-based classifier produced more specific semantic types: "Sunrise" rather than "Time" recognized by the content-based classifier, or "Dewpoint" rather than "Temperature." At other times, especially when the service uses novel terms in parameter names, the content-based classifier outperforms the metadata-based classifier. Therefore, combining them may improve the accuracy of the output parameter labeling. We tested this hypothesis by using the metadata-based classifier to predict semantic types of the output parameters. The classifier's top prediction was correct in 145 cases ($A = 0.68$). Sixty one of the incorrect top-rated guesses were attributed to the "Unknown" class. This suggests a strategy of combining results of metadata and content-based classification to improve the performance of the semantic labeling algorithm. Namely, we take the most likely prediction produced by the metadata-based classifier as the semantic type of the output parameter, unless it happens to be "Unknown." In that case, we use the top prediction of the content-based classifier. If content-based

[3]Our services included commercial services, such as Arcweb and DEVPRIM geolocation services, DOTS and EJSE weather services, and a handful of free online services.

[4]These data come previous queries to known sources which have been stored in a local database.

Table 2: Input and output parameter classification results

classifier	total	correct	accuracy
	(a) intput parameters		
metadata-based	47	43	0.91
	(b) output parameters		
content-based	213	107	0.50
metadata-based	213	145	0.68
combined	213	171	0.80

algorithm did not assign that output parameter to any semantic types, we look at the second-rated prediction of the metadata-based classifier. The combined approach correctly classifies 171 output parameters ($A = 0.80$), significantly outperforming individual classifiers (Table 2).

Conclusion

We presented the problem of automatically learning semantic types of data used by Web services. This is an important part of semantic modeling of Web services to enable them to be automatically integrated by information mediators. We described metadata-based classification algorithm, that can be used to assign semantic types to the service's input parameters. This algorithm represents each parameter by a set of features — terms extracted from the service's WSDL file. To test the classifier's predictions, we invoked the service with sample data of the predicted type. If the prediction was correct, the Web service returned output data. We then used content-based classifier to assign semantic types to the outputs. We evaluated performance of both classification algorithms on a variety of domains. Next, we validated our approach by semantically labeling several Web services in the Weather and Geospatial information domains. Combined performance of the two classifiers was very good, showing that they can be used to accurately and automatically label data used by Web services.

There remain many ways to improve our algorithms. We would like to improve the performance of metadata-based classifier in multi-input parameter domains by considering dependencies between input parameters. We also need to improve the performance of the content-based classifier on complex data types, and data expressed in different formats. Meta-analysis of classification results can help improve them, by eliminating duplicates, for example. Our future research will address these topics.

Acknowledgements

This research is based by work supported in part by the NSF under Award Nos. IIS-0324955 and CNS-0509517, in part by the Air Force Office of Scientific Research under grant number FA9550-04-1-0105, in part by DARPA under Contract No. NBCHD030010. We would like to thank ArcWeb for providing us with access to their data.

The U.S.Government is authorized to reproduce and distribute reports for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any of the above organizations or any person connected with them.

References

Baker, D. L., and McCallum, K. A. 1998. Distributional clustering of words for text classification. In *Proc. of ACM SIG on Information Retrieval (SIGIR-1998)*.

Carman, M., and Knoblock, C. A. 2005. Learning Source Descriptions for Web Services. In *Proc. of AAAI05 Workshop on Exploring Planning and Scheduling for Web Services, Grid and Autonomic Computing*.

Doan, A.; Domingos, P.; and Halevy, A. Y. 2001. Reconciling schemas of disparate data sources: A machine-learning approach. In *Proc. of ACM SIG on Management of Data (SIGMOD-2001)*, 509–520.

Doan, A.; Domingos, P.; and Halevy, A. 2003. Learning to match the schemas of databases: A multistrategy approach. *Machine Learning Journal* 50:279–301.

Dong, X.; Halevy, A.; Madhavan, J.; Nemes, E.; and Zhang, J. 2004. Similarity search for web services. In *Proceedings of the International Conference on Very Large Databases (VLDB-2004)*.

Hess, A., and Kushmerick, N. 2003. Learning to attach semantic metadata to web services. In *Proceedings 2nd International Semantic Web Conference (ISWC2003)*.

Lerman, K.; Getoor, L.; Minton, S.; and Knoblock, C. A. 2004. Using the Structure of Web Sites for Automatic Segmentation of Tables. In *Proc. of ACM SIG on Management of Data (SIGMOD-2004)*.

Lerman, K.; Gazen, C.; Minton, S.; and Knoblock, C. A. 2004. Populating the Semantic Web. In *Proc. of AAAI04 Workshop on Advances in Text Extraction and Mining*.

Lerman, K.; Minton, S.; and Knoblock, C. 2003. Wrapper maintenance: A machine learning approach. *Journal of Artificial Intelligence Research* 18:149–181.

Li, W., and Clifton, C. 2000. Semint: A tool for identifying attribute correspondence in heterogeneous databases using neural networks. *Data and Knowledge Engineering* 33:49–84.

Minka, T. P. 2003. A comparison of numerical optimizers for logistic regression. http://research.microsoft.com/∼minka/papers/logreg/.

Mitchell, T. 2005. *Machine Learning*. 2nd edition (draft) edition. Chapter 1: Generative and Discriminative Classifiers: Nave Bayes and Logistic Regression.

Ng, A. Y., and Jordan, M. I. 2002. On discriminative vs. generative classifiers: A comparison of logistic regression and nave bayes. In *Proc. of Neural Information Processing Systems (NIPS02)*.

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

Thakkar, S.; Ambite, J. L.; and Knoblock, C. A. 2005. Composing, optimizing, and executing plans for bioinformatics web services. *VLDB Journal* 14(3):330–353.