Machine Learning for Annotating Semantic Web Services

Andreas Heß and **Nicholas Kushmerick**
Computer Science Department, University College Dublin, Ireland
{andreas.hess, nick}@ucd.ie

Introduction

Emerging Semantic Web standards promise the automated discovery, composition and invocation of Web Services. Unfortunately, this vision requires that services describe themselves with large amounts of hand-crafted semantic metadata.

We are investigating the use of machine learning techniques for semi-automatically classifying Web Services and their messages into ontologies. From such semantically-enriched WSDL descriptions, it is straightforward to generate significant parts of a service's description in OWL-S or a similar language.

In this paper, we first introduce an application for annotating Web Services that is currently under development. Our application reads Web Service descriptions from WSDL files and assists the user in annotating them with classes and properties from an ontology written in OWL. Our ultimate goal is a semi-automated approach that makes use of the machine learning algorithms described in this paper to suggest classifications to the user, thereby simplifying the creation of semantic metadata.

Second, we describe the application of the well known Naive Bayes and SVM algorithms to the task of Web Service classification. We show that an ensemble approach that treats Web Services as structured objects is more accurate than an unstructured approach.

Third, we discuss the dependencies between a Web Service's category and its operations and input and output messages. We briefly sketch possibilities and challenges for the classification task that arise when considering these depencies.

Finally, we describe a probabilistic algorithm for learning to assign semantic labels to input parameters. Because Web Services were not widely available at the time when these experiments were done, we looked at HTML forms instead. However, the described techniques are relevant to Web Services.

Three Semantic Taxonomies

We begin by elaborating on what we mean by semantic metadata. To automatically invoke a particular Web Service, metadata is needed to facilitate (at a minimum) the discovery that a particular operation of some particular Web Service is appropriate, as well as the semantic meaning of each operation and each input/output parameter. For example, to invoke an operation that queries an airline's timetable, the service must be annotated with metadata indicating that the operation does indeed relate to airline timetable querying, and each parameter must be annotated with the kind of data that should be supplied (departure data, time and airport, destination airport, return date, number of passengers, etc).

The goal of our research is to develop algorithms for classifying (according to some agreed ontology) a Web Service, each of its operations, and each operations' input and output messages. In particular, we assume three ontologies for attaching semantic metadata to Web Services.

- First, we assume a *category* taxonomy $\mathcal{C}$. The category of a Web Service describes the general kind of service that is offered, such as "services related to travel", "information provider" or "business services".
- Second, we assume a *domain* taxonomy $\mathcal{D}$. Domains capture the purpose of a specific service operation, such as "searching for a book", "finding a job", "querying a airline timetable", etc.
- Third, we assume a *datatype* taxonomy $\mathcal{T}$. Datatypes relate not to low-level encoding issues such as "string" or "integer", but to the expected semantic category of a field's data, such as "book title", "salary", "destination airport", etc.

We do not advocate a new semantic language, such as an alternative to OWL-S.Instead, we believe that the issues that are addressed by our research are more generic and we do not commit to any particular standard.

Web Service Annotation Tool

The goal of our research is to enable users to semi-automatically annotate Web Services. We believe that fully automated discovery and invocation is still quite far in the future and that for several reasons it will be desirable that a human is still in the loop. Also, unlike approaches such as OWL-S that aim for full semantic description, we keep our approach simple by focusing only on the purpose, inputs and ouputs of a service. We ignore preconditions and effects as well as composition aspects and restrict the view to discovery and incovation. A screenshot of the application

we are developing is shown in Fig. 1. In our tool a WSDL description is parsed and the user is shown the list of operations and XML Schema types. The user can assign classes and properties in an ontology to the Web Service itself and its operations, parameters and complex types. The ontology can be created externally and imported using DAML+OIL or OWL.

The intended user for this application is not necessarily a Web Service creator, but rather an administrator that operates a hub of Web Services such as a UDDI registry or a user wanting to integrate and invoke several Web Services.

Initially, we have used this tool to generate training data for our learning algorithms. Eventually, our learning algorithms will sit "behind" our tool's user interface, automatically suggesting probable classifications.

Relations to OWL-S

However, our approach can also complement existing efforts from the Semantic Web community. Our application will complement tools such as the "WSDL to OWL-S tool" (Paolucci *et al.* 2003), which generates an OWL-S skeleton from a WSDL-described Web Service. Note that this tool deals only with the parts of OWL-S that can be extracted directly from the WSDL. Moreover, the semantic annotations we generate would allow the Web Service to be placed into a profile hierarchy. Our tool also provides the mapping from XSD types to concepts. This will complement tools that create XSL for use in the OWL-S Grounding out of such a mapping, like the DL-XML Mapper Workbench (Peer 2003). However, note that our annotation does not address the issue of defining composite processes.

Web Service Category Classification

As described in (Heß & Kushmerick 2003), we treat the determination of a Web Service's category as a text classification problem, where the text comes from the Web Service's WSDL description. In some of the experiments we have also used plain text descriptions that occur in a UDDI registry or on a Web page where the Web Service is described. Unlike standard texts, WSDL descriptions are highly structured. Our experiments demonstrate that selecting the right set of features from this structured text improves the performance of a learning classifier.

To extract terms for our text classification algorithms, we parsed the port types, operations and messages from the WSDL and extracted names as well as comments from various `<documentation>` tags. We did not extract standard XML Schema data types or informations about the service provider. The extracted terms from the WSDL as well as the terms that came from the plain text descriptions were stemmed with Porter's algorithm, and a stop-word list was used to discard low-information terms.

Data sets

We used two different data sets for our experiments. The first data set is the same that we used in (Heß & Kushmerick 2003). For this data set, we gathered a corpus of 424 Web Services from SALCentral.org, a Web Service index.

Category taxonomy $\mathcal{C}$ and number of services for each category

BUSINESS (22)	COMMUNICATION (44)	CONVERTER (43)
COUNTRY INFO (62)	DEVELOPERS (34)	FINDER (44)
GAMES (9)	MATHEMATICS (10)	MONEY (54)
NEWS (30)	WEB (39)	*discarded* (33)

Domain taxonomy $\mathcal{D}$ and number of forms for each domain

SEARCHBOOK (44)	FINDCOLLEGE (2)	SEARCHCOLLEGEBOOK (17)
QUERYFLIGHT (34)	FINDJOB (23)	FINDSTOCKQUOTE (9)

Datatype taxonomy $\mathcal{T}$ (illustrative sample)

Address	NAdults	Airline	Author
BookCode	BookCondition	BookDetails	BookEdition
BookFormat	BookSearchType	BookSubject	BookTitle
NChildren	City	Class	College
CollegeSubject	CompanyName	Country	Currency
DateDepart	DateReturn	DestAirport	DestCity
Duration	Email	EmployeeLevel	...

Figure 2: Categories $\mathcal{C}$, domains $\mathcal{D}$ and datatypes $\mathcal{T}$.

An assistant manually classified these Web Services into a taxonomy $\mathcal{C}$, see Fig. 2. This person, a research student with no previous experience with Web Services, was advised to adaptively create new categories herself and was allowed to arrange the categories as a hierarchy. However, we used only the 25 top level categories. We then discarded categories with less than seven instances, leaving 391 Web Services in 11 categories that were used in our experiments. The discarded Web Services tended to be quite obscure, such as a search tool for a music teacher in an area specified by ZIP code.

The second data set was created later with our annotation tool. In contrast to the old data set, where only the categories were annotated, a full annotation of all operations and input and output messages was done. We selected 164 Web Services and arranged them within a hierarchy with only 5 top level categories (26 categories in total), forming a less skewed data set in order to gather a large set of similar services. The category ontology is shown in Fig. 3. We created the category ontology ourselves. Four final year students annotated the Web Services's operations, messages and types over a period of four days. During that time, the domain and datatype ontologies evolved and the students created new classes and properties as needed.

From our second data set we used only the category annotation for the experiments in this section. We describe preliminary experiments with the domain and datatype annotation in the next section. However, unlike the taxonomy from the first dataset, we used not only the 5 top level categories for our classification but all 23 classes in the ontology that were used for annotating services. Three classes were present in the ontology as upper classes for other classes, but no service was annotated with them.

Experiments

With both datasets we experimented with four bags of words each, denoted by *A*–*D*. The combination of bags of words is marked in Fig. 4. We also used combinations of these bags of words, where e.g. *C+D* denotes a bag of words

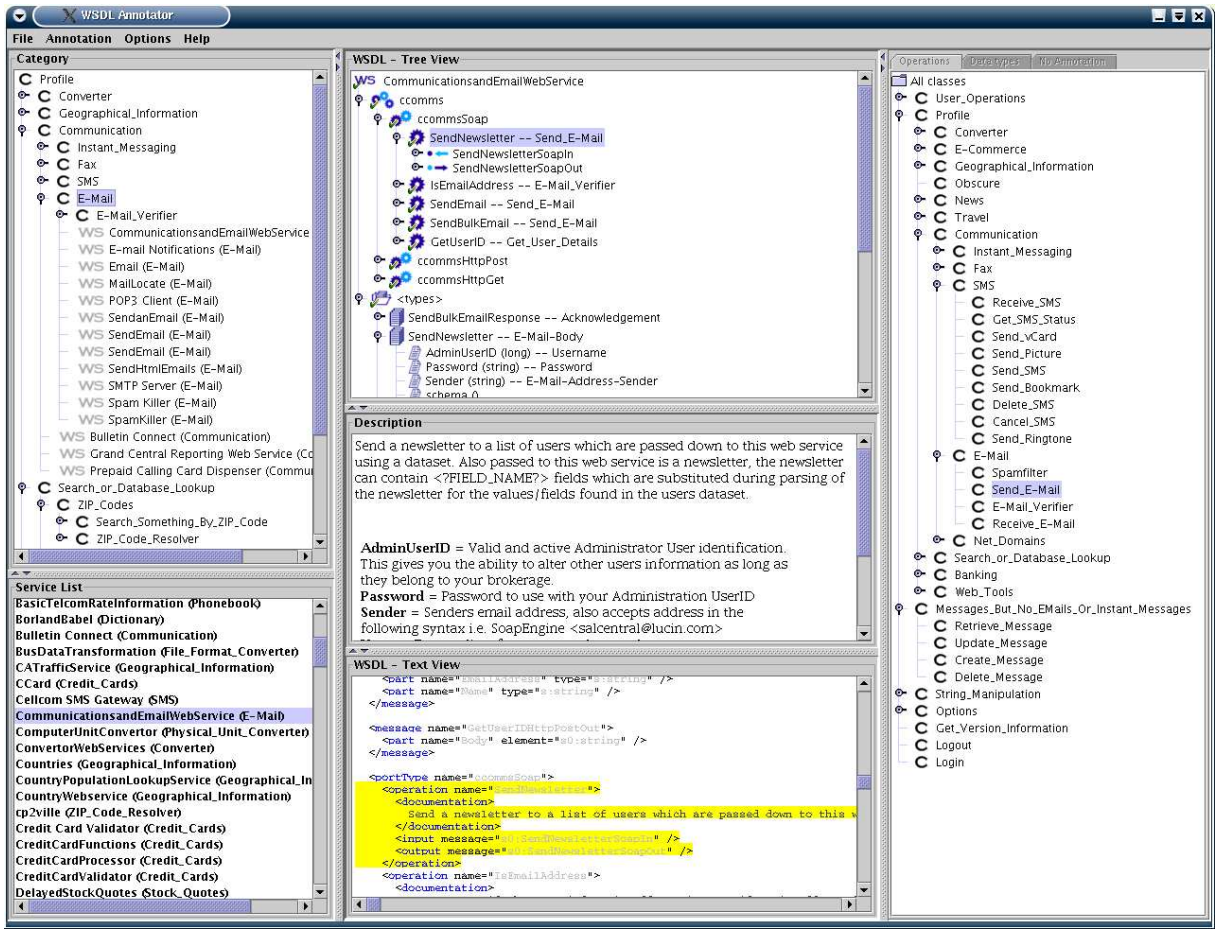

Figure 1: Screenshot of our annotation application.

that consists of the descriptions of the input and output messages. We converted the resulting bag of words into a feature vector for supervised learning algorithms, with attributes weighted based on simple term frequency. We experimented with more sophisticated TFIDF-based weighting schemes, but they did not improve the results.

As learning algorithms, we used the Naive Bayes, SVM and HyperPipes algorithms as implemented in Weka (Witten & Frank 1999). In our experiments the Naive Bayes algorithm was generally used in a multi-class setup using a one-against-all scheme. We combined several classifiers in an ensemble learning approach. Ensemble learners make a prediction by voting together the predictions of several "base" classifiers and are a well known machine learning technique, e.g. (Dietterich 2000). Ensemble learning has been shown in a variety of tasks to be more reliable than the base classifiers: the whole is often greater than the sum of its parts. To combine two or more classifiers, we multiplied the confidence values obtained from the multi-class classifier implementation. For some settings, we tried weighting of these values as well, but this did not improve the overall performance. We denote a combination of different algorithms or different feature sets by slashes, e.g. Naive Bayes(*A/B+C+D*) denoting two Naive Bayes classifiers, one trained on the plain text description only and one trained one all terms extracted from the WSDL.

We split our tests into two groups. First, we tried to find the best split of bags of words using the terms drawn from the WSDL only (bags of words *B–D*). These experiments are of particular interest, because the WSDL is usually automatically generated (except for the occasional comment tags), and the terms that can be extracted from that are basically operation and parameter names. The results for the experiments with the data from the WSDL only are shown in Fig. 5. Note that we did not use any transmitted data, but only the parameter descriptions and the XML schema. Second, we look how the performance improves, if we include the plain text description (bag of words *A*). The results for these experiments are shown in Fig. 6. The values shown in these two diagrams were obtained with the first dataset.

Evaluation

We evaluated the different approaches using a leave-one-out method. Our results show that the "obvious" approach of using one big bag of words that contains everything (i.e. *A+B+C+D* for WSDL and descriptions, or *B+C+D* for the

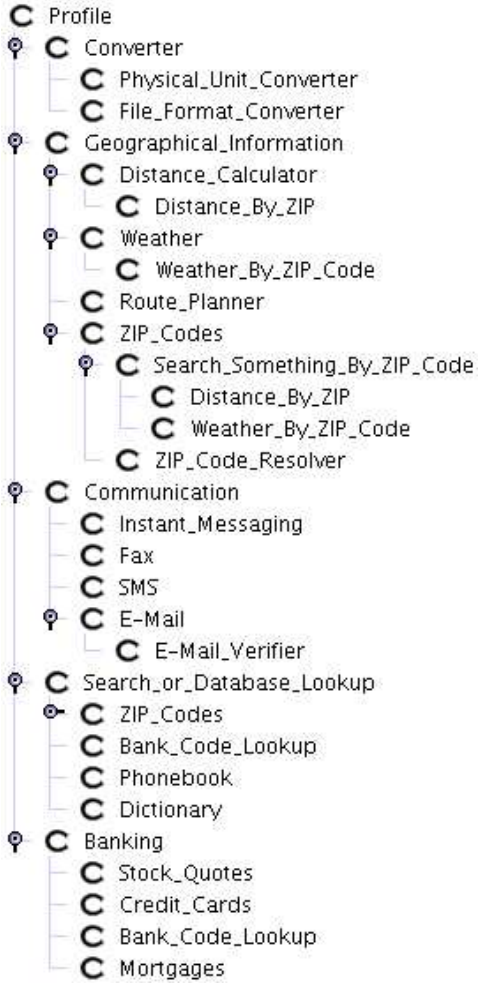

Figure 3: Categories $\mathcal{C}$ for the second data set

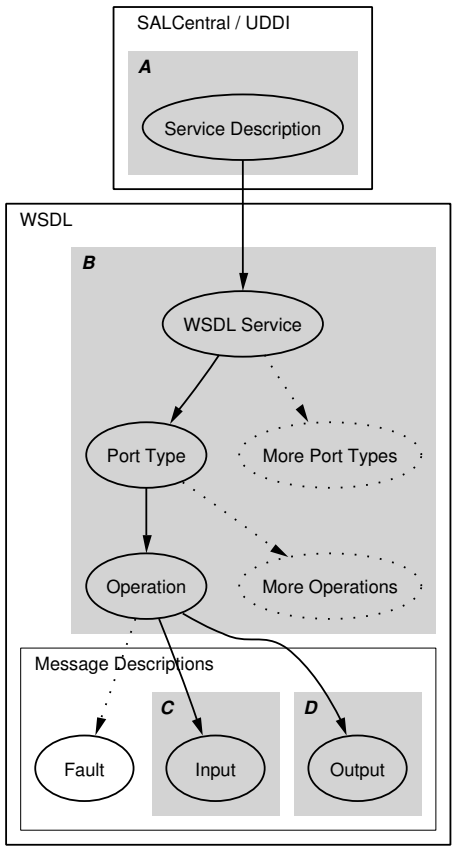

Figure 4: Text structure for our Web Service corpus.

WSDL-only tests) generally performs worst. These classifiers do not perform better than classifiers that are trained on only one of the B, C or D bags of words. We included these classifiers in Figs. 5, 6, 7 and 8 as baselines.

Ensemble approaches where the bags of words are split generally perform better. This is intuitive, because we can assume a certain degree of independence between for example the terms that occur in the plain text descriptions and the terms that occur in the WSDL description.

The A bag of words containing the plain text description of the service is an exeption. A classifier trained on this bag of words alone performs significantly better than any single classifier trained on one of the other bags of words. This is also intuitive, because we can assume that a plain text description of a Web Service's capabilities contains more information than for example a list of its operation names.

When is Ensemble Learning appropriate?

Our experiments with the second dataset show that ensemble learning is most appropriate when we can create different views on a learning problem that lead to classifiers that yield about the same performance each.

When we look at the performance of a SVM classifier that only uses one of the B, C or D bags of words, we find out that each individual classifier has a performance of between 38% and 44% accuracy, as shown in Fig. 7. When we combine these three classifiers in an ensemble as described above we increase the accuracy to 50%.

If we assume that each of the three individual classifiers is affected by random noise then it is straightforward to see that any ensemble based on voting will improve the result, because the noise rarely effects all three views on the same instance. Thus, an error in one of the views is levelled out by the two other views.

Having this in mind, it is also clear why this approach does not always work well when one single view outperforms the other views. In our example, this is the case for the Naive Bayes classifier that uses the A bag of words, shown in Fig. 8.

A Naive Bayes classifier using only this view achieves an accuracy of over 64%. Any ensemble that combines this view with other views that performs worse leads to a reduction of performance. Again, when we assume that the output of the classifiers is affected by a noise function, we see an explanation for this effect: If some of the classifiers are significantly more affected by noise, then voting together two classifiers will not level out the noise, but rather will the noise inadvertently affect the classifier that by itself would perform better.

This effect can also be seen in the first dataset, although not as strong as in the second. In the first dataset, a Naive Bayes classifier using only the plain text description scores only slightly worse than the best ensemble. In the second dataset, a Naive Bayes classifier using only the plain text performs even better than any ensemble.

However, this is not a strict rule. As shown in Fig. 7, the performance of the SVM ensemble classifier still increases even if we add the plain text descriptions, although also for SVM a single classifier trained on the A bag of words is more accurate than any other single classifier.

Results

A user would also save a considerable amount of work if he or she only had to choose between a small number of predicted categories. For this reason, we also report the accuracy when we allow near misses. Figs. 5 and 6 show how the classifiers improve when we increase this tolerance threshold. For our best classifier, the correct category is in the top 3 predictions 82% of the time.

Classifiying Web Service Domains and Datatypes

A fundamental assumption behind our work is that there are interdependencies between a Web Service's category, and the domains and datatypes of its operations. For example, a Web Service in the "services related to travel" category is likely to support an operation for "booking an airline ticket", and an operation for "finding a job" is likely to require a "salary requirement" as input. Relational learning algorithms, such as those compared in (Neville & Jensen 2003), address this problem. In our current and future work, we are developing an algorithm to combine evidence from all sources and make predictions for all three taxonomies at the same time. Preliminary experiments suggest that the domain of an operation and the datatypes of its inputs and outputs can be classified as accurately as the categories.

Domain Classification

We tested an SVM classifier on the operations and their domains from our dataset. This dataset contains 1138 instances in 136 classes. Although 136 classes is a rather high number for a text classification task and although most of the operations are undocumented and thus the operation name is the only text source, the classifier performed astonishingly well.

We evaluated the classifier using a 10-fold cross-validation scheme. The accuracy is 86.9% and the macro-averaged F1-measure is 0.75. Note that the classifier in this experiment performs much better than the category classifier, although one might think that the problem is much harder.

Datatype Classification

We also tested a SVM classifier on the datatypes from a Web Service's inputs and outputs. Due to the large number of 1854 instances and 312 classes we evaluated these classifiers by using a percentage-split method. We split our dataset randomly into a 66% training set and a 34% test set. The SVM classifier achieved an accuracy of 62.14%, but a macro-averaged F1-measure of only 0.34. The reason for the low F1 is that the data set is highly skewed. Many Web Services require a "Username" and a "Password" as input, while only very few Web Services require a "Weather Station Code". We believe, however, that this is a problem that can effectively be addressed by exploiting the relations between domains and datatypes. An operation from the "Query Weather" domain is very likely to require a "Weather

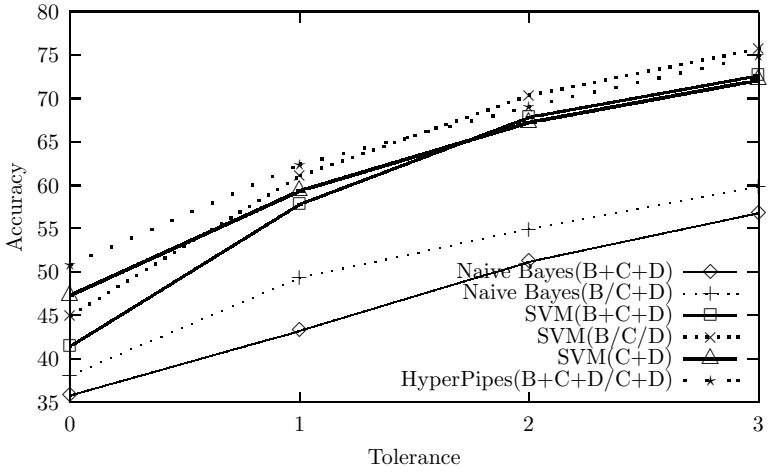

Figure 5: Classification accuracy for WSDL only, dataset 1

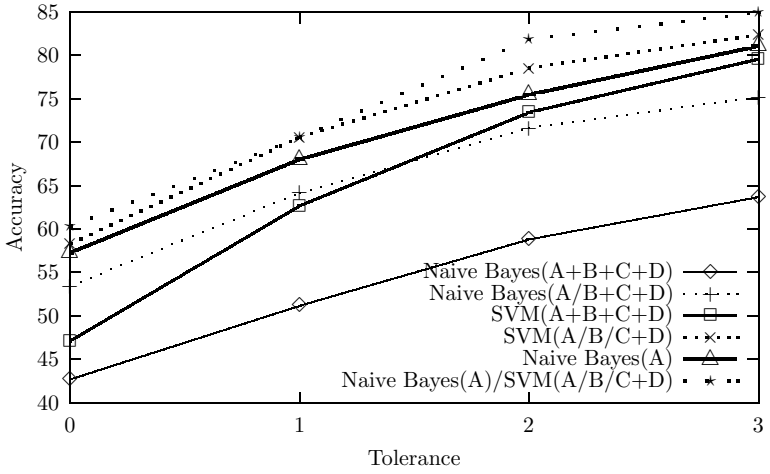

Figure 6: Classification accuracy WSDL and descriptions, dataset 1

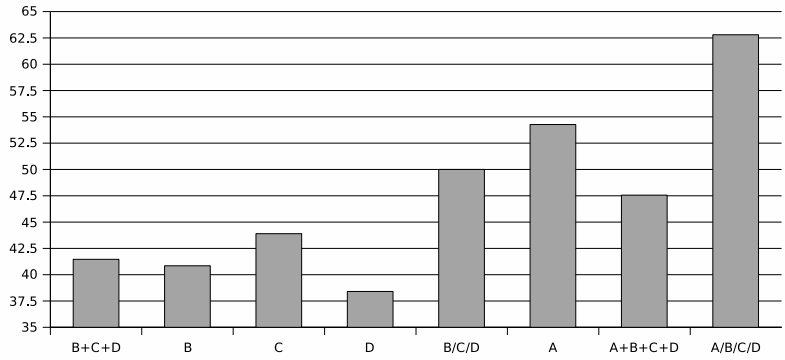

Figure 7: Classification accuracy for SVM, dataset 2

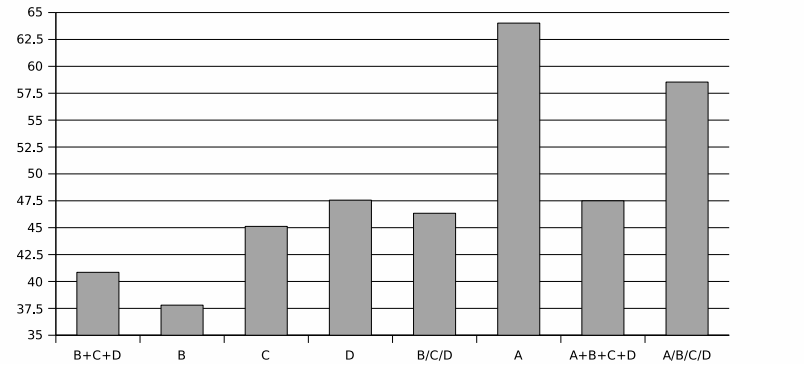

Figure 8: Classification accuracy for Naive Bayes, dataset 2

Station Code" as input, while an operation from another domain will almost certainly not. The next two sections will explain this idea in greater detail.

Using the Domain to Classify the Category

We carried out a preliminary experiment to test if we can exploit the dependency between domain and category in a direct way. We trained a classifier on the domains of a Web Service's operations and let it predict the service's category. We evaluated the result using the leave-one-out method. The Naive Bayes classifier trained on this data achieved an accuracy of 86.0% (a SVM classifier achieved 73.1%). Note that a Naive Bayes classifier that is trained on the textual description of a Web Service achieves only a 64.0% accuracy (SVM 54.3%). A Web Service's operation's domains are thus a better indicator to the Web Service's category than a plain text description.

Although of course the operation's domains are usually unknown as well as the category, this experiment suggests that the dependencies between domain and category can indeed be exploited in a simple manner. Our future work is to explore this area. We treat the task as a multi-view learning problem where the different views are interconnected. We are currently developing an iterative approach to learning a classifier for interconnected multi-view tasks.

The basic idea is that the result of the classification of the domain will, in the next iteration, affect the classification of the category and the datatypes, and vice-versa.

Having seen that a classifier trained on the domains is a better predictor for the category than a classifier trained on the category's data itself, we believe that this approach is very promising.

Web Form Classification

As described above, we have not yet fully explored the potential of the depencies between domain and datatype with our new dataset. However, in our older experiments with Web forms (Kushmerick 2003), we exploited this connection by using a Bayesian network as illustrated in Fig. 9. A Bayesian Network is a causal graph. The edges of the graph indicate conditional probabilities between entities, or the flow of evidence. The learning task is to estimate the parameters of the stochastic generative model from a set of training data.

Given such a Bayesian network, classifying a form involves setting the probability for each term and then computing the maximum-likelihood form domain and field datatypes consistent with that evidence.

We have evaluated our approach using a collection of 129 Web forms comprising 656 fields in total, for an average of 5.1 fields/form. As shown in Fig. 2, the domain taxonomy $\mathcal{D}$ used in our experiments contains 6 domains, and the datatype taxonomy $\mathcal{T}$ comprises 71 datatypes.

The forms were manually gathered by browsing Web forms indices such as InvisibleWeb.com for relevant forms. Each form was then inspected by hand to assign a domain to the form as a whole, and a datatype to each field.

To extract the terms for the classification algorithm the raw HTML was postprocessed in various ways. Roughly, terms occuring in the HTML were associated with the nearest input field in the form. Note that this step may generate noisy training data that would not affect the algorithm if applied to Web Services.

For domain prediction, our algorithm has an F1 score of 0.87 while the baseline scores 0.82. For datatype prediction, our algorithm has an F1 score of 0.43 while the baseline scores 0.38. We conclude that our "holistic" approach to form and field prediction is more accurate than a greedy baseline approach of making each prediction independently.

Discussion

Future Work

We are currently extending our classification algorithms in several directions. Our approaches ignore valuable sources of evidence—such as the actual data passed to/from a Web Service—and it would be interesting to incorporate such evidence into our algorithms. Our algorithms could be extended in a number of ways, such as using statistical methods such as latent semantic analysis as well as thesauri like WordNet.

We envision a single algorithm that incorporates the category, domain, datatype and term evidence. To classify all the operations and inputs of a Web Service at the same time, a Bayesian network like the one in Fig. 9 could be constructed

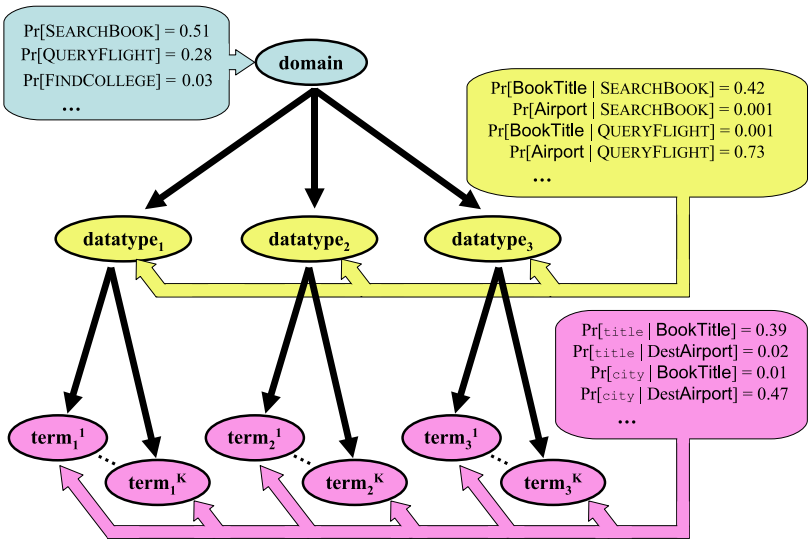

Figure 9: The Bayesian network used to classify a Web form containing three fields.

for each operation, and then a higher-level category node could be introduced whose children are the domain nodes for each of the operations.

Ultimately, our goal is to develop enabling technologies that could allow for the semi-automatic generation of Web Services metadata. We would like to use our techniques to develop a toolkit that emits metadata conforming to Semantic Web standards such as OWL-S.

Conclusions

The emerging Web Services protocols represent exciting new directions for the Web, but interoperability requires that each service be described by a large amount of semantic metadata "glue". We have presented approaches to automatically generating such metadata, and evaluated our approaches on a collection of Web Services and forms.

Although we are far from being able to automatically create semantic metadata, we believe that the methods we have presented here are a reasonable first step. Our preliminary results indicate that some of the required semantic metadata can be semi-automatically generated using machine learning techniques.

Acknowledgments. This research is supported by grants SFI/01/F.1/C015 from Science Foundation Ireland, and N00014-03-1-0274 from the US Office of Naval Research.

References

Dietterich, T. G. 2000. Ensemble methods in machine learning. *Lecture Notes in Computer Science* 1857.

Heß, A., and Kushmerick, N. 2003. Learning to attach semantic metadata to Web Services. In *Proc. Int. Semantic Web Conf.*

Kushmerick, N. 2003. Learning to invoke Web forms. In *Proc. Int. Conf. Ontologies, Databases and Applications of Semantics*.

Neville, J., M. R., and Jensen, D. 2003. Statistical relational learning: Four claims and a survey. In *Proceedings of the Workshop on Learning Statistical Models from Relational Data, 8th International Joint Conference on Artificial Intelligence*.

Paolucci, M.; Srinivasan, N.; Sycara, K.; and Nishimura, T. 2003. Towards a semantic choreography of web services: From WSDL to DAML-S. In *International Conference for Web Services*.

Peer, J. 2003. DL-XML Mapper Workbench. http://sws.mcm.unisg.ch/xmldl/manual.html.

Witten, I. H., and Frank, E. 1999. *Data Mining: Practical machine learning tools with Java implementations*. San Francisco: Morgan Kaufmann.