Learning to Attach Semantic Metadata to Web Services

Andreas Heß and Nicholas Kushmerick

Computer Science Department, University College Dublin, Ireland
{andreas.hess,nick}@ucd.ie

Abstract. Emerging Web standards promise a network of heterogeneous yet interoperable Web Services. Web Services would greatly simplify the development of many kinds of data integration and knowledge management applications. Unfortunately, this vision requires that services describe themselves with large amounts of semantic metadata "glue". We explore a variety of machine learning techniques to semi-automatically create such metadata.
We make three contributions. First, we describe a Bayesian learning and inference algorithm for classifying HTML forms into semantic categories, as well as assigning semantic labels to the form's fields. These techniques are important as legacy HTML interfaces are migrated to Web Services. Second, we describe the application of the 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. Finally, we describe a clustering algorithm that automatically discovers the semantic categories of Web Services. All of our algorithms are evaluated using large collections of real HTML forms and Web Services.

1 Introduction

Emerging Web standards such as WSDL [w3.org/TR/wsdl], SOAP [w3.org/TR/soap], UDDI [uddi.org] and DAML-S [www.daml.org/services] promise an ocean of Web Services, networked components that can be invoked remotely using standard XML-based protocols. For example, significant e-commerce players such as Amazon and Google export Web Services giving public access to their content databases.

The key to automatically invoking and composing Web Services is to associate machine-understandable semantic metadata with each service. A central challenge to the Web Services initiative is therefore a lack of tools to (semi-)automatically generate the necessary metadata. We explore the use of machine learning techniques to automatically create such metadata from training data. Such an approach complements existing uses of machine learning to facilitate the Semantic Web, such as for information extraction [7,9,3] and for mapping between heterogeneous data schemata [5].

The various Web Services standards involve metadata at various levels of abstraction, from high-level advertisements that facilitate indexing and matching

D. Fensel et al. (Eds.): ISWC 2003, LNCS 2870, pp. 258–273, 2003.

relevant services, to low-level input/output specifications of particular operations. The various metadata standards are evolving rapidly, and the details of current standards are beyond the scope of this paper. Rather than committing to any particular standard, we investigate the following three sub-problems, which are essential components to any tool for helping developers create Web Services metadata.

1. To automatically invoke a particular Web Service operation, metadata is needed to indicate the overall "domain" of the operation, as well as the semantic meaning of each of the operation's input parameters. For example, to enable automatic invocation of a Web Service operation that queries an airline's timetable, the operation 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 date, time and airport, destination airport, return date, number of passengers, etc). In Sec. 2, we propose to *automatically assign a Web Form to a concept in a domain taxonomy, and to assign each input parameter to a concept in a data-type taxonomy.*
2. A Web Service is a collection of operations, and Web Services must be grouped into coherent "categories" of services supporting similar operations. For example, many airlines may each export a Web Service that supports similar operations such as querying for flights, checking whether a flight is delayed, checking a frequent-traveller account balance, etc. To enable the retrieval of appropriate Web Services, in Sec. 3 we describe techniques to *automatically assign a Web Service to a concept in a category taxonomy.*
3. Finally, when Web Services are widely deployed, it may well be infeasible to agree a category taxonomy in advance. We therefore propose in Sec. 4 to *cluster Web Services in order to automatically create a category taxonomy.*

Fig. 1 describes the relationship between the category, domain and datatype taxonomies that motivate our research. In more detail, our work can be characterized in terms of the following three levels of metadata. 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.

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. The structure of Fig. 1 indicates the main ways in which these constraints inter-

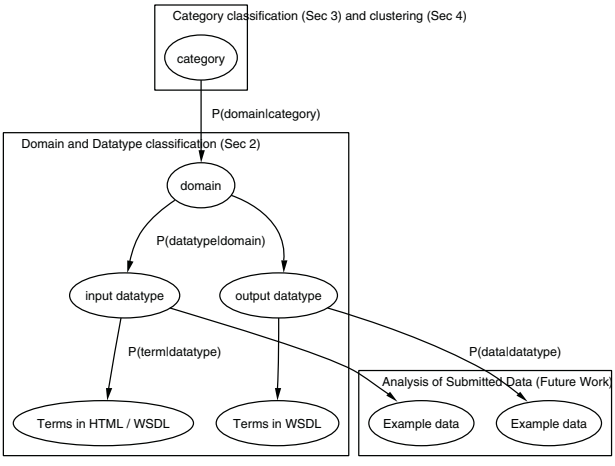


Fig. 1. A Web Service's category is dependent on the domains and datatypes of its operations.

act. The edges of the graph indicate conditional probabilities between entities, or the flow of evidence.

Finally, the boxes in Fig. 1 indicate the actual algorithms that we describe in this paper. As indicated above, in Sec. 2 focuses on the domain and datatype taxonomies, while Secs. 3–4 focus on the category taxonomy. Note that we do not exploit the additional evidence that the category gives us for the classification at the domain and datatype level. As part of our future work, we intend to exploit this connection, as well as additional evidence (e.g., the data actually sent to/from the Web Service).

2 Supervised Domain and Datatype Classification

We begin by describing an algorithm for classifying HTML forms into semantic categories, as well as assigning semantic labels to each form field. These techniques are important as legacy HTML interfaces are migrated to Web Services.

2.1 Problem Formulation

Web form instances are structured objects: a form comprises one or more fields, and each field in turn comprises one or more terms. More precisely, a *form* F_i is a sequence of *fields*, written $F_i = [f_i^1, f_i^2, \ldots]$, and each field f_i^j is a bag of *terms*, written $f_i^j = [t_i^j(1), t_i^j(2), \ldots]$.

The Web form learning problem is as follows. The input is set of labeled forms and fields; that is, a set $\{F_1, F_2, \ldots\}$ of forms together with a domain $D_i \in \mathcal{D}$ for each form F_i, and a datatype $T_i^j \in \mathcal{T}$ for each field $f_i^j \in F_i$. The

output is a form classifier; that is, a function that maps an unlabeled form F_i, to a predicted domain $D_i \in \mathcal{D}$, and a predicted datatype $T_i^j \in \mathcal{T}$ for each field $f_i^j \in F_i$.

We assume a domain taxonomy $\mathcal{D}$ and a datatype taxonomy $\mathcal{T}$. We use SmallCaps to indicate domains, so we might have $\mathcal{D} = \{$SearchBook, FindJob, QueryFlight, $\ldots\}$. SansSerif style indicates datatypes, so we might have $\mathcal{T} = \{$BookTitle, Salary, DestAirport, $\ldots\}$.

2.2 Generative Model

Our solution to the Web form classification is based on a stochastic generative model of a hypothetical "Web service designer" creating a Web page to host a particular service. First, the designer first selects a domain $D_i \in \mathcal{D}$ according to some probability distribution $\Pr[D_i]$. For example, in our experiments, forms for finding books were quite frequent relative to forms for finding colleges, so $\Pr[$SearchBook$] \gg \Pr[$FindCollege$]$.

Second, the designer selects datatypes $T_i^j \in \mathcal{T}$ appropriate to D_i, by drawing from some distribution $\Pr[T_i^j|D_i]$. For example, presumably $\Pr[$BookTitle$|$ SearchBook$] \gg \Pr[$DestAirport$|$SearchBook$]$, because services for finding books usually involve a book's title, but rarely involve airports. On the other hand, $\Pr[$BookTitle$|$ QueryFlight$] \ll \Pr[$DestAirport$|$QueryFlight$]$.

Finally, the designer writes the Web page that implements the form by coding each field in turn. More precisely, for each selected datatype T_i^j, the designer uses terms $t_i^j(k)$ drawn according to some distribution $\Pr[t_i^j(k)|T_i^j]$. For example, presumably $\Pr[$`title`$|$BookTitle$] \gg \Pr[$`city`$|$BookTitle$]$, because the term `title` is much more likely than `city` to occur in a field requesting a book title. On the other hand, presumably $\Pr[$`title`$|$DestAirport$] \ll \Pr[$`city`$|$DestAirport$]$.

2.3 Parameter Estimation

The learning task is to estimate the parameters of the stochastic generative model from a set of training data. The training data comprises a set of N Web forms $\mathcal{F} = \{F_1, \ldots, F_N\}$, where for each form F_i the learning algorithm is given the domain $D_i \in \mathcal{D}$ and the datatypes T_i^j of the fields $f_i^j \in F_i$.

The parameters to be estimated are the domain probabilities $\hat{\Pr}[D]$ for $D \in \mathcal{D}$, the conditional datatype probabilities $\hat{\Pr}[T|D]$ for $D \in \mathcal{D}$ and $T \in \mathcal{T}$, and the conditional term probabilities $\hat{\Pr}[t|T]$ for term t and $T \in \mathcal{T}$. We estimate these parameters based on their frequency in the training data: $\hat{\Pr}[D] = N_{\mathcal{F}}(D)/N$, $\hat{\Pr}[T|D] = M_{\mathcal{F}}(T, D)/M_{\mathcal{F}}(D)$, and $\hat{\Pr}[t|T] = W_{\mathcal{F}}(t, T)/W_{\mathcal{F}}(T)$, where $N_{\mathcal{F}}(D)$ is the number of forms in the training set $\mathcal{F}$ with domain D; $M_{\mathcal{F}}(D)$ is the total number of fields in all forms of domain D; $M_{\mathcal{F}}(T, D)$ is the number of fields of datatype T in all forms of domain D; $W_{\mathcal{F}}(T)$ is the total number of terms of all fields of datatype T; and $W_{\mathcal{F}}(t, T)$ is the number of occurrences of term t in all fields of datatype t.

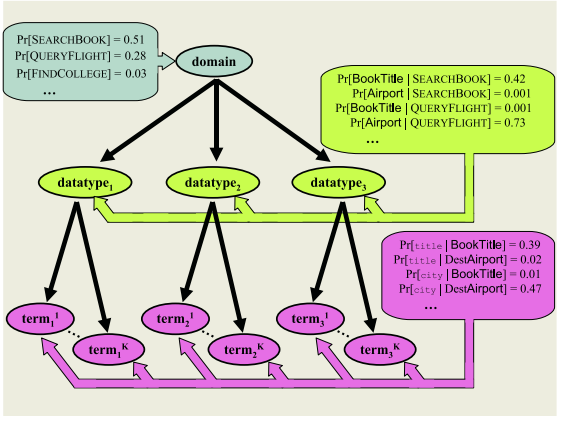


Fig. 2. The Bayesian network used to classify a Web form containing three fields.

2.4 Classification

Our approach to Web form classification involves converting a form into a Bayesian network. The network is a tree that reflects the generative model: a root node represents the form's domain, children represent the datatype of each field, and grandchildren encode the terms used to code each field.

In more detail, a Web form to be classified is converted into a three-layer tree-structured Bayesian network as follows. The first (root) layer contains just a single node **domain** that takes on values from the set of domains $\mathcal{D}$. The second layer consists of one child $\mathbf{datatype}_i$ of **domain** for each field in the form being classified, where each $\mathbf{datatype}_i$ take on values from the set $\mathcal{T}$.

The third (leaf) layer comprises a set of children $\{\mathbf{term}_i^1, \ldots, \mathbf{term}_i^K\}$ for each $\mathbf{datatype}_i$ node, where K is the number of terms in the field. The term nodes take on values from the vocabulary set $\mathcal{V}$, defined as the set of all terms that have occurred in the training data.

Fig. 2 illustrates the network that would be constructed for a form with three fields and K terms for each field. (Each field contains the same number K of terms/field for simplicity; in fact, the number of term nodes reflects the actual number of terms in the parent field.)

The conditional probability tables associated with each node correspond directly to the learned parameters mentioned earlier. That is, $\Pr[\mathbf{domain} = D] \equiv \hat{\Pr}(D)$, $\Pr[\mathbf{datatype}_i = T | \mathbf{domain} = D] \equiv \hat{\Pr}(T|D)$, and $\Pr[\mathbf{term}_i^k = t | \mathbf{datatype}_i = T] \equiv \hat{\Pr}(t|T)$. Note that the conditional probabilities tables are identical for all datatype nodes, and for all term nodes.

Given such a Bayesian network, classifying a form $F_i = [f_i^1, f_i^2, \ldots]$ involves "observing" the terms in each field (i.e., setting the probability $\Pr[\mathbf{term}_i^k = t_i^j(k)] \equiv 1$ for each term $t_i^j(k) \in f_i^j$), and then computing the maximum-likelihood form domain and field datatypes consistent with that evidence.

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	...

Fig. 3. Subsets of the domain and datatype taxonomies used in the experiments.

2.5 Evaluation

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. 3, 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 manually 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.

After the forms were gathered, they were segmented into fields. We discuss the details below. For now, it suffices to say that we use HTML tags such as `<input>` and `<textarea>` to identify the fields that will appear to the user when the page is rendered. After a form has been segmented into fields, certain irrelevant fields (e.g., submit/reset buttons) are discarded. The remaining fields are then assigned a datatype.

A final subtlety is that some fields are not easily interpreted as "data", but rather indicate minor modifications to either the way the query is interpreted, or the output presentation. For example, there is a "help" option on one search services that augments the requested data with suggestions for query refinement. We discarded such fields on a case-by-case basis; a total of 12.1% of the fields were discarded in this way.

The final data-preparation step is to convert the HTML fragments into the "form = sequence of fields; field = bag of terms" representation. The HTML is first parsed into a sequence of tokens. Some of these tokens are HTML field tags (e.g., `<input>`, `<select>`, `<textarea>`). The form is segmented into fields by associating the remaining tokens with the nearest field. For example, "`<form> a <input name=f1> b c <textarea name=f2> d </form>`" would be segmented as "`a <input name=f1> b`" and "`c <textarea name=f2> d`".

The intent is that this segmentation process will associate with each field a bag of terms that provides evidence of the field's datatype. For example, our classification algorithm will learn to distinguish labels like "Book title" that are associated with BookTitle fields, from labels like "Title (Dr, Ms, ...)" that indicate PersonTitle.

Finally, we convert HTML fragments like "`Enter name: <input name=name1 type=text size=20> <br>`" that correspond to a particular field, into the field's bag of terms representation. We process each fragment as follows.

First, we discard HTML tags, retaining the values of a set of "interesting" attributes, such as an `<input>` tag's `name` attribute. The result is "`Enter name: name1`". Next, we tokenize the string at punctuation and space characters, convert all characters to lower case, apply Porter's stemming algorithm [11], discard stop words, and insert a special symbol encoding the field's HTML type (text, select, radio-button, etc). This yields the token sequence [`enter`, `name`, `name1`, **TypeText**]. Finally, we apply a set of term normalizations, such as replacing terms comprising just a single digit (letter) with a special symbol **SingleDigit** (**SingleLetter**), and deleting leading/trailing numbers. In this example the final result is the sequence [`enter`, `name`, `name`, **TypeText**].

2.6 Results

We begin by comparing our approach to two simple bag of terms baselines using a leave-one-out methodology. For domain classification, the baseline uses a single bag of all terms in the entire form. For datatype classification, the baseline approach is the naive Bayes algorithm over its bag of terms.

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.

While our approach is far from perfect, we observe that form classification is extremely challenging, due both to noise in the underlying HTML, and the fact that our domain and datatype taxonomies contain many classes compared to traditional (usually binary!) text classification tasks.

While fully-automated form classification is our ultimate goal, an imperfect form classifier can still be useful in interactive, partially-automated scenarios in which a human gives the domain or (some of) the datatypes of a form to be labelled, and the classifier labels the remaining elements.

Our first experiment measures the improvement in datatype prediction if the Bayesian network is also provided as evidence the form's domain. In this case our algorithm has an F1 score of 0.51, compared to 0.43 mentioned earlier.

Our second experiment measures the improvement in domain prediction if evidence is provided for a randomly chosen fraction α of the fields' datatypes, for $0 \leq \alpha \leq 1$. $\alpha = 0$ corresponds to the fully automated situation in which no datatype evidence is provided; $\alpha = 1$ requires that a person provide the datatype of every field. As shown in Fig. 4, the domain classification F1 score increases rapidly as α approaches 1.

Our third investigation of semi-automated prediction involves the idea of ranking the predictions rather than requiring that the algorithm make just one prediction. In many semi-automated scenarios, the fact that the second- or third-ranked prediction is correct can still be useful even if the first is wrong. To formalize this notion, we calculate F1 based on treating the algorithm as correct if the true class is in the top R predictions as ranked by posterior probability. Fig. 4 shows the F1 score for predicting both domains and datatypes, as a

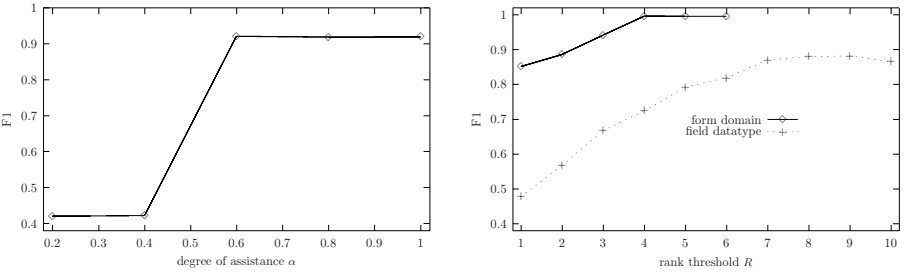


Fig. 4. F1 as a function of (left) the fraction α of field datatypes supplied by the user, and (right) the rank threshold R.

function of R. $R = 1$ corresponds to the cases described so far. We can see that relaxing R even slightly results in a dramatic increase in F1 score.

So far we have assumed unstructured datatype and domain taxonomies. However, domains and datatypes exhibit a natural hierarchical structure (e.g., "forms for finding something" vs. "forms for buying something"; or "fields related to book information" vs. "fields related to personal details"). It seems reasonable that in partially-automated settings, predicting a similar but wrong class is more useful than a dissimilar class.

To explore this issue, our research assistants converted their domain and datatype taxonomies into trees, creating additional abstract nodes to obtain reasonable and compact hierarchies. We used distance in these trees to measure the "quality" of a prediction, instead of a binary "right/wrong". For domain predictions, our algorithm's prediction is on average 0.40 edges away from the correct class, while the baseline algorithm's predictions are 0.55 edges away. For datatype prediction, our algorithm's average distance is 2.08 edges while the baseline algorithm averages 2.51. As above, we conclude that our algorithm outperforms the baseline.

3 Supervised Category Classification

The previous section addressed the classification of Web forms and their fields. We now address how to categorize Web Services. Since Web Services can export more than one operation, a Web Service corresponds loosely to a set of Web forms. As described in Sec. 1, we are therefore interested in classifying Web Services at the higher *category* level ("Business", "Games", etc.), rather than the lower *domain* level ("search for a book", "purchase a book", etc.) used for classifying Web forms.

3.1 Problem Formulation

We assume a set $\mathcal{C} = \{C_1, C_2, \ldots\}$ of Web Service categories. Each C_i corresponds formally to a subset of some domain ontology $\mathcal{D}$: $C_i \in 2^{\mathcal{D}}$ for each i.

Category taxonomy $\mathcal{C}$ and number of Web 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)

Fig. 5. Web Service categories $\mathcal{C}$.

For example, the "Business" category would include any Web Service whose operations are related in some way to business.

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. 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. By combining different classifiers it is possible to improve the performance even further, although for both the feature selection and the combination no general rule exists.

In the following sections, we describe our Web Service corpus, describe the methods we used for classification, and evaluate our approach.

3.2 Web Services Corpus

We gathered a corpus of 424 Web Services from SALCentral.org, a Web Service index. These Web Services were then manually classified into a hierarchical taxonomy $\mathcal{C}$. To avoid bias, the person was a research student with no previous experience with Web Services. The person has the same information as given on SALCentral.org, and was allowed to inspect the WSDL description if necessary. The person was advised to adaptively create new categories while classifying the Web Services and was allowed to arrange the categories as a hierarchy.

The 424 Web Services were classified by our assistant into 25 top level categories. As shown in Fig. 5, we then discarded categories with less than seven instances, leaving 391 Web Services in eleven 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. Even for a human classifier, these services would be extremely hard to classify. Note that the distribution after discarding these classes is still highly skewed, ranging from nine Web Services in the "Games" category, to 62 services in the "Country Information" category.

3.3 Ensemble Learning

As shown in Fig. 6, the information available to our categorization algorithms comes from two sources. First, the algorithms use the Web Service description in the WSDL format, which is always available to determine a service's category. Second, in some cases, additional descriptive text is available, such as from a UDDI entry. In our experiments, we use the descriptive text provided by

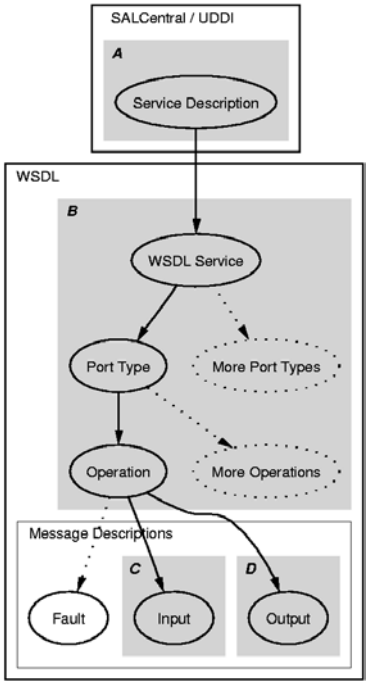


Fig. 6. Text structure for our Web Service corpus

SALCentral.org, since UDDI entries were not available. We parse the port types, operations and messages from the WSDL and extract names as well as comments from various "documentation" tags. We do not extract standard XML Schema data types like string or integer, or informations about the service provider. The extracted terms are stemmed with Porter's algorithm, and a stop-word list is used to discard low-information terms.

We experimented with four bags of words, denoted by A–D. The composition of these bags of words is marked in Fig. 6. We also used combinations of these bags of words, where e.g. $C+D$ denotes a bag of words 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 terms weighted based on simple 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 [15]. We combined several classifiers in an ensemble learning approach. Ensemble learners make a prediction by voting together the predictions of several "base" classifiers. 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. 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).

3.4 Evaluation

We evaluated the different approaches using a leave-one-out methodology.

Our results show that using a classifier with one big bag of words that contains everything (i.e. $A+B+C+D$ for WSDL and descriptions, or $B+C+D$ for the WSDL-only tests) generally performs worst. We included these classifiers in Fig. 7 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. What is a bit more surprising is that for some settings we achieve very good results if we use only a subset of the available features, i.e. only one of the bags of words. So, in these cases, sometimes one part is greater than the whole. However, we could not find a generic rule for how to best split the available bags of words, as this seems to be strongly dependent on the algorithm and the actual data set.

Space restrictions prevent us from showing all our results, so in Fig. 7 we give the accuracy for the classifiers that performed best. Note that SVM generally performs better than Naive Bayes, except for the classifier where we used the plain text descriptions only. An ensemble consisting of three SVM classifiers performs good for both the WSDL-only setting and also when including the descriptions. However, the best results are achieved by other combinations.

In a machine learning setting with a split feature set it is also possible to use Co-Training [1] to improve classification accuracy, if unlabeled data is present. In preliminary experiments we added 370 unlabeled Web Services. In this particular setting we could gain no advantage using Co-Training, but due to time restrictions we were not able to fully explore this area.

For a semi-automatic assignment of the category to a Web Service, it is not always necessary that the algorithm predicts the category exactly, although this is of course desirable. A human developer would also save a considerable amount of work if he or she only had to choose between a small number of categories. For this reason, we also report the accuracy when we allow near misses. Fig. 7 shows how the classifiers improve when we increase this tolerance threshold. For our best classifier, the correct class is in the top 3 predictions 82% of the time.

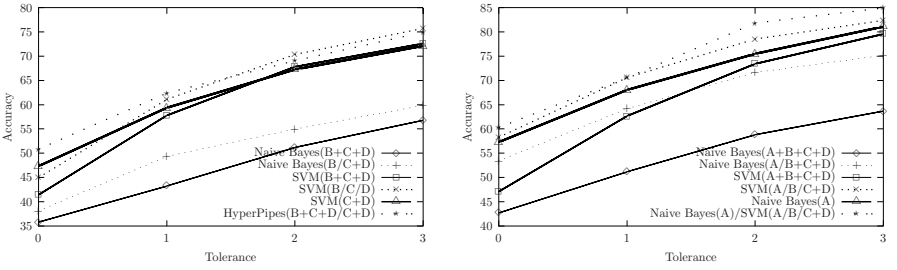


Fig. 7. Classification accuracy for WSDL only (left), and both WSDL and descriptions (right).

4 Unsupervised Category Clustering

As a third approach towards our goal of automatically creating Web Services metadata, we explored the use of unsupervised clustering algorithms to automatically discover the semantic categories of a group of Web Services.

4.1 Clustering Algorithms

We tested five clustering algorithms on our collection of Web Services. First, we tried a simple k-nearest-neighbour algorithm. Hierarchical group average and complete link algorithms (e.g. [14,12]) serve as representatives of traditional approaches. We also tried a variant of the group average clusterer that we call Common-Term, and the Word-IC algorithm [16]. The Word-IC algorithm, unlike the other clustering algorithms, does not rely on a traditional cosine based similarity measure between the documents (i.e. in the web services in our case), but hierarchically merges clusters based on the number of intersecting words and a global quality function. The global quality function also serves as a halting criterion. The Common Term algorithm halts, when the top level clusters do not share any terms. For the group average and complete link algorithms, we used a minimum similarity between documents as a halting criterion. As a baseline, we partition the Web Services into eleven random clusters.

Our Common-Term algorithm differs from the standard group average clustering in the way the centroid document vector is computed. Instead of using all terms from all the sub-clusters, only the terms that occur in all sub-clusters form the centroid are used. Like the Word-IC algorithm, our hope is that this leads to short and concise cluster labels.

We use the standard cosine approach with TFIDF weighting to measure the similarity $\sigma(a, b)$ between two "documents" a and b.

4.2 Quality Metrics for Clustering

Evaluating clustering algorithms is a task that is considerably harder than evaluating classifications, because we cannot always assign a cluster to a certain

reference class. Several quality measures have been proposed; see [13] for a recent survey.

We have evaluated our clustering algorithms using Zamir's quality function [16], and the normalized mutual information quality metric described in [13]. We also introduce a novel measure inspired by the well-known precision and recall metrics that correlates well with other quality measures and has a simple probabilistic interpretation.

In the literature on evaluating clustering algorithms, precision and recall have only been used on a per-class basis. This assumes that a mapping between clusters and reference classes exists. The fraction of documents in a cluster that belong to the "dominant" class, i.e. the precision assuming the cluster corresponds to the dominant class, is known as purity. Usage of the purity measure is problematic, if the cluster contains an (approximately) equal number of objects from two or more classes. This clustering might not even be unintuitive, if it is merely the case that the granularity of the clustering is coarser than that of the reference classes.

We modify the definitions of precision and recall to consider *pairs* of objects, rather than individual objects. Let n be the number of objects. Then there are $\frac{n(n-1)}{2}$ pairs of objects. Each such pair must fall into one of four categories: the objects are put in the same class by both the reference clusters and the clustering algorithm, they are clustered together but in difference reference clusters, etc.

clustered together?		yes	no
in same reference class?	yes	a	b
	no	c	d

If a, b, c and d are the number of object pairs in each case, then $a+b+c+d = \frac{n(n-1)}{2}$. Precision and recall can now be computed the same way as in standard information retrieval: precision $= a/(a+c)$ and recall $= a/(a+b)$. Other metrics such as F1 or accuracy are defined in the usual way.

Note that there is a simple probabilistic interpretation of these metrics. Precision is equivalent to the conditional probability that two documents are in the same reference class given they are in the same cluster. Recall is equivalent to the conditional probability that two documents are in the same cluster given they are in the same reference class. Let $\mathcal{R}$ denote the event that a document pair is in the same reference class and $\mathcal{C}$ denote that a document pair is in the same cluster. Then we have that precision $= P(\mathcal{R} \wedge \mathcal{C} \mid \mathcal{C}) = P(\mathcal{R} \mid \mathcal{C})$, and recall $= P(\mathcal{R} \wedge \mathcal{C} \mid \mathcal{R}) = P(\mathcal{C} \mid \mathcal{R})$.

Note that precision is biased towards small clusters, but because we are considering document pairs it is not trivially maximized by placing every document in its own cluster. Recall is biased against large clusters, as it reaches the maximum when all documents are placed in one cluster. Finally, we observe that precision and recall are symmetric, in the sense that precision$(A, B) =$ recall(B, A) for any two clusterings A and B.

4.3 Evaluation

Fig. 8 shows the precision and F1 scores of the clusters generated by the various algorithms we tried. We do not report Zamir's quality measure $Q(\mathrm{C})$ or the normalized mutual information measure $Q(\mathrm{NMI})$, because they are highly correlated with our precision metric. Precision is biased towards a large number of small clusters, because it is easier for a clusterer to find a small number of similar services multiple times than to find a large number of similar clusters. Therefore we believe that also $Q(\mathrm{C})$ and $Q(\mathrm{NMI})$ are in fact biased towards small clusters, although it is claimed that $Q(\mathrm{NMI})$ is not biased, and although they do not reach their optimal value for singleton clusters.

Not surprisingly, none of the algorithms does particularly well, because the Web Services clustering problem is quite challenging. In many cases even humans disagree on the correct classification. For example, SALCentral.org manually organized its Web Services into their own taxonomy, and their classification bears little resemblance to ours. Furthermore, we have 11 categories in our reference classification, which is a rather high number. However, in terms of precision, all our algorithms outperform the random baseline.

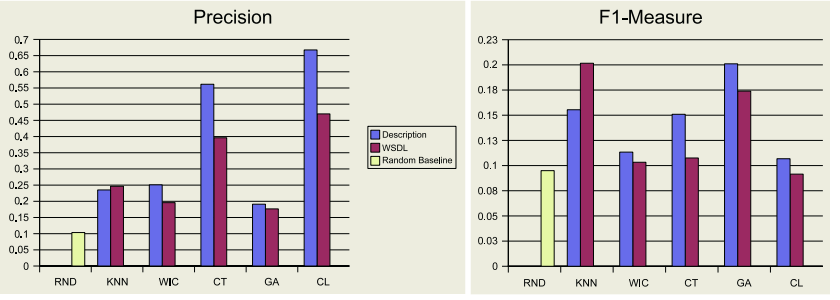
All clustering algorithms tend to "over-refine", meaning that they produce far more clusters than classes exist in the reference data. For the group average and complete-link algorithms, the number of clusters could be decreased if we set a lower minimum similarity, but especially the group average algorithm then tends to produce very large clusters with more than 100 Web Services.

Note that recall is strongly affected by such an over-refinement. In our experiments, it turns out that the F1 (see Fig. 8), is largely dominated by recall. However, precision is more important than recall in the application scenarios that motivate our research. Specifically, consider generic automatic data integration scenarios in which Web Services that have been clustered together are then automatically invoked simultaneously. For example, a comparison shopping agent would invoke operations from all Web Services that were clustered into a "E-Commerce" category. Precision is more important than recall for the same reason why precision is more important in today's document search engines: in a Web populated by millions of Web Services, the danger is not that a relevant service will be missed, but that irrelevant services will be inadvertently retrieved.

We conclude from these data that Web Service category clustering is feasible based just on WSDL descriptions, through clearly hand-crafted text descriptions (e.g., SALCentral.org's description or text drawn from UDDI entries) produce even better results.

5 Discussion

Future Work. We are currently extending our classification and clustering algorithms in several directions. Our approaches ignore a valuable sources of evidence—such as the actual data passed to/from a Web Service—and it would



RND: Random baseline, KNN: k-Nearest-Neighbour, WIC: Word-IC, CT: Common Term, GA: Group Average, CL: Complete Link

Fig. 8. Precision and F1 for the various clustering algorithms.

be interesting to incorporate such evidence into our algorithms. Our clustering algorithm 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 shown in Fig. 1. To classify all the operations and inputs of a Web Service at the same time, a Bayesian network like the one in Fig. 2 could be constructed 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 DAML/DAML-S.

Related Work. There has been some work on matching of Web Services (e.g. [10,2]), but they require manually-generated explicit semantic metadata.

Clustering is a well-known technique, although it has not yet been applied to Web Services. Besides the traditional group-average or single-link approaches, newer algorithms like Word-IC or the Scatter/Gather-algorithms [4] exist.

The search capabilities of UDDI are very restricted, though various extensions are available or under development (e.g. UDDIe [www.cs.cf.ac.uk/user/A.Shaikhali/uddie]). Kerschberg et al are planning to apply the techniques they introduced in WebSifter [6] to UDDI.

When we actually want to simultaneously invoke multiple similar Web Services and aggregate the results, we encounter the problem of XML schema mapping (e.g., [5,8]).

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 three approaches to automatically generating such metadata, and evaluated our approach on a collection of Web Services and forms.

Although we a 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 requisite semantic metadata can be semi-automatically generated using machine learning, information retrieval and clustering techniques.

Acknowledgments. This research was 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

1. Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *COLT: Proceedings of the Workshop on Computational Learning Theory, Morgan Kaufmann Publishers*, 1998.
2. J. Cardoso. *Quality of Service and Semantic Composition of Workflows*. PhD thesis, Department of Computer Science, University of Georgia, Athens, GA, 2002.
3. F. Ciravegna. Adaptive information extraction from text by rule induction and generalization. In *17th Int. Joint Conference on Artifical Intelligence*, 2001.
4. D. Cutting, J. Pedersen, D. Karger, and J. Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. In *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 318–329, 1992.
5. A. Doan, P. Domingos, and A. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *Proc. SIGMOD Conference*, 2001.
6. L. Kerschberg, W. Kim, and A. Scime. Intelligent web search via personalizable meta-search agents. pages 1345–1358, 2002.
7. N. Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118(1–2):15–68, 2000.
8. S. Melnik, H. Molina-Garcia, and E. Rahm. Similariy flooding: A versatile graph matching algorithm. In *Proc. of the International Conference on Data Engineering (ICDE), 2002*.
9. I. Muslea, S. Minton, and C. Knoblock. A Hierachical Approach to Wrapper Induction. In *Proc. 3rd Int. Conf. Autonomous Agents*, pages 190–197, 1999.
10. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matchmaking of web services capabilities. In *Int. Semantic Web Conference*, 2002.
11. M.F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.
12. G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.
13. Alexander Strehl. *Relationship-based Clustering and Cluster Ensembles for High-dimensional Data Mining*. PhD thesis, University of Texas, Austin, 2002.
14. C.J. van Rijsbergen. *Information Retrieval*. Butterworths, London, 2nd edition, 1979.
15. Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools with Java implementations*. Morgan Kaufmann, San Francisco.
16. Oren Zamir, Oren Etzioni, Omid Madani, and Richard M. Karp. Fast and intuitive clustering of web documents. In *Knowledge Discovery and Data Mining*, pages 287–290, 1997.