Ontology-Based Categorization of Web Services with Machine Learning

Adam Funk and Kalina Bontcheva
Department of Computer Science
University of Sheffield
Regent Court, 211 Portobello
S1 4DP, Sheffield, UK
{a.funk,k.bontcheva}@dcs.shef.ac.uk

Abstract. We discuss the problem of categorizing web services according to a shallow ontology for presentation on a specialist portal. We treat it as a text classification problem and apply first information extraction techniques (using keywords and rules), then machine learning. The techniques are evaluated according to standard measures for flat categorization as well as the Balanced Distance Metric for ontological classification and compared with related work in web service categorization. The results for machine learning are very good and will let us take advantage of the portal’s Web 2.0 features to improve it.

Key words: web services, semantic annotation, machine learning, service classification, Web 2.0

1 Introduction

The web is no longer just a set of documents; services now feature prominently, and both research and industry devote considerable effort and interest to service-oriented architecture (SOA), web services, and related technologies. In a web of services or a service-oriented-architecture, discovery is an essential task for building and using applications. Unfortunately the widely used service description techniques only cover the syntactic level. Previous research projects showed that semantic descriptions (such as OWL-S, WSMO, and SAWSDL) can enable precise service discovery, but they are not widely deployed [1].

The main existing solution for service discovery is UDDI (Universal Description, Discovery and Integration), a standard for programmatically publishing and retrieving structured information about web services. Some companies operate public UDDI repositories, but its success has not been widespread because of complicated registration, missing monitoring facilities, and other difficulties. A few portals have been designed to act as web service repositories, but they all rely on manual registration and review, so coverage is limited and information falls out of date easily. Furthermore, the standard web search engines do not

1 http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm
provide effective ways to search for services or to allow filtering according to availability and service parameters.

The Service-Finder project aims to address this problem for a wider audience by offering a comprehensive framework for service discovery through a portal\textsuperscript{2}, so we must address the problem of creating semantic descriptions of Web Services. The TAO project\textsuperscript{3} dealt with semi-automatic creation of semantic service descriptions, but focused on legacy applications and relied on the existence of substantial software documentation. Service-Finder offers automatic creation of service descriptions for a wide range of publicly available services and enables service consumers (not just providers) to collaboratively enrich the semantic descriptions according to Web 2.0 principles.

2 The research problem

The information flow in Service-Finder is structured as follows. The Service Crawler (SC) carries out focused crawling for web services and archives WSDL files and related HTML files, then passes batches of these data to the Automatic Annotator (AA), which processes them using information extraction techniques to produce semantic annotations for the Conceptual Indexer and Matcher (CIM). The CIM acts as a semantic repository and back end for the Service-Finder Portal (SFP)—it answers the latter’s queries and records users’ annotations in the repository along with the automatic annotations. The Clustering Engine (CE) also provides recommendations by clustering users and services based on the users’ behaviour and interests [2, 1].

This paper focuses on a specific problem in Service-Finder: automatically categorizing web services so that portal users can find them effectively. Although users can improve the category annotations, the aim is to provide reasonably good ones to start with so users will find the portal useful and contribute—so they do not have to face 23 000 uncategorized services with only keyword searching.

2.1 Data

A batch of AA input from the SC consists of several files in the Heritrix\textsuperscript{4} Internet Archive format; Table 1 summarizes a recent batch, which we call Crawl 3 (the results of the SC’s crawl in June 2009). The SC also provides an index which associates each document with one service identifier. (The services and providers are identified by URIs, and a service’s URI can be easily converted to the relevant provider’s URL.)

The AA’s task is to extract information (such as contact addresses), classify documents and services, and semantically annotate the services, providers, and

\textsuperscript{2}http://demo.service-finder.eu/
\textsuperscript{3}http://www.tao-project.eu/
\textsuperscript{4}http://crawler.archive.org/
interesting documents, using tools built on the GATE [3] infrastructure. This paper presents work on the service categorization task.

Table 1 also shows the results of preprocessing the archive files to produce datastores of GATE documents (relating to about 8 000 providers and 23 000 services). The preprocessor merges exact duplicate documents\(^5\), suppresses WSDL documents that the XML parser rejects, and suppresses documents whose provider URIs indicate that the services are not usable from the crawled WSDL files because they contain invalid endpoints (they have been incorrectly generated or the endpoints have been deliberately edited out of the publicly available versions).

The AA uses the WSDL documents to generate instances of Endpoint, Interface, and Operation, and to instantiate relations between them and the Service instances, but carries out very little textual information extraction (IE) on them. The abstracts are compiled by the SC from various textual elements and attributes in the WSDL files (service name, documentation, operation names, input and output parameters, etc.). The SC generates one abstract for each service URI, choosing the best WSDL file associated with the service (typically the shortest WSDL URL containing the provider name). The AA carries out textual IE on the abstracts as well as the HTML files.

### 2.2 The categorization task

The category ontology used in the project contains 59 subclasses of Category for web services, such as Genetics, Address Information, and Media Management, arranged in a shallow tree (down to three levels below the top class) in seven

---

\(^5\) Each provider is associated with one or more services. Each input document is associated by the SC with one service, but merged documents that refer to different services (of the same provider) are associated with all of them. Each abstract relates to only one service.
main branches (such as Business, Consumer, and Science). Each service can be annotated with more than one category. (See [1, 4] for further details.)

Since we originally had no manually annotated data to use for training but needed to generate category annotations quickly for the first release of the portal, we devised a weighted voting system based on keywords; this was relatively easy to integrate with the AA pipeline which was already being developed to apply gazetteers and rules to the documents for other purposes. §3.1 explains this approach in more detail and presents the evaluation results.

After the first release of the portal, members of the project manually annotated 224 services through the portal to produce 387 category annotations (using 45 of the service categories), which we exported to use for evaluation of the keyword-based system and as training data for machine learning (ML). At that point, we did not have enough annotations from outside users to make a significant contribution, but in the future, their annotations will also be exported and used in the same way. §3.2 explains and evaluates the ML techniques used.

3 Methodology

3.1 Keywords, rules, and voting

For our first approach to categorization, we devised an ad hoc gazetteer of keywords and phrases associated with service categories (the category names, parts of the multiword names, morphological variations (such as plurals), synonyms, and related words obtained by examining the documents); Table 2 shows examples for three of the categories. This gazetteer processor was configured to be case-insensitive but to match whole words only (for example, Address would match part of address list but nothing in headdress).
Table 3. Evaluation of keyword-based service categorization (limited to two categories per service)

<table>
<thead>
<tr>
<th>Crawl</th>
<th>traditional %</th>
<th>BDM %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P</td>
<td>R</td>
</tr>
<tr>
<td>Crawl 1</td>
<td>35.8</td>
<td>16.3</td>
</tr>
<tr>
<td>Crawl 2</td>
<td>39.3</td>
<td>12.4</td>
</tr>
<tr>
<td>Crawl 3</td>
<td>20.0</td>
<td>31.6</td>
</tr>
</tbody>
</table>

Because some place where keywords can appear are more significant than others, JAPE rules\(^6\) later in the pipeline create special annotations over the gazetteer matches, each of which has a multiplier \(m\) which starts at 1 and is increased up to 5 if the keyword occurs in an HTML \(<\text{title}>\) or heading element, a \(<\text{p}>\) element with a significant number of interesting keywords (from this and other gazetteers), or an interesting\(^7\) document.

Each category keyword match in a document is later treated as \(m\) (the multiplier) votes for the relevant category for each service associated with the document. These votes are compiled for every service, which is labelled with the two highest-scoring categories. (This arbitrary limit was agreed within the project.)

We used the manually annotated data described in §2.2 to evaluate the AA’s categorization of those services. Table 3 shows the traditional IE measures (precision, recall, and \(F_1\) [6]) for the three major datasets (which we call Crawls 1, 2, and 3) that have been processed through Service-Finder, as well as the Balanced Distance Metric (BDM) [7] for the latest one. The traditional measures count each categorization as wrong if it is not an exact match, even if the key (manual) and response (automatic) categories are close in the ontology tree. BDM scoring, however, counts 1 for an exact match but gives partial credit for getting a superclass or subclass of the key, as well as for hitting another node on the same branch of the class tree. It counts a response as 0 only if its lowest superclass in common with the key is the top class. These augmented precision, recall, and \(F_1\) measures are more suitable for ontology-based classification [8].

The BDM scores for Crawl 3 are noticeably higher than the traditional ones; this indicates that the system is categorizing quite a few services with “nearly correct” classes from the category ontology. These results are still not very good, however. After Crawl 2, we improved recall by adding more keywords to the gazetteers, but this made precision deteriorate significantly. We had suspected that the documents in other languages than English might pose a problem, but we added an automatic language identification tool based on character \(n\)-grams [9] to count the number of abstracts and HTML documents by language,

\(^6\) JAPE processors in GATE compile rules that specify regular expressions of annotations and create additional annotations, manipulate features, and execute Java code when matches are detected. [5]

\(^7\) The IE pipeline adjusts each document’s interesting feature between 0 and 3 according to the information it finds.
as shown in Table 4 and concluded that adding translations to the gazetteers would not be fruitful. Instead, we pursued experiments in service categorization based on machine learning.

3.2 Machine learning-based Service Categorization

We treat this as a text classification problem and use the Support Vector Machine (SVM) technique, which is well documented for this purpose. [10–12] SVM text classification is carried out using n-grams of features of sequential units of text (such as unmodified tokens, part-of-speech tags, orthographic features, or lemmata), but previous research indicates that increasing n from 1 to 2 slows down performance and rarely improves the results, and increasing n beyond 2 typically deteriorates the results. [11, 13, 14]

One disadvantage of SVM ML is that it can learn and apply only one class to each instance (document, in this case), so we had to simplify the problem for services with more than one manually annotated category by using the lowest (most specific) class in the category ontology (or making an arbitrary choice between two equal categories). As we shall see later, however, it is possible to generate more than one category for a service with multiple documents.

We carried out several document classification experiments on the manually annotated services using standard parameters for GATE’s SVM tool, with unigrams of tokens as the instance attributes (i.e., treating each document as a bag of words) and one service category per document as the target class, one-against-others classification, and a linear kernel. (See [11, 12] for the technical details of the parameters.) All the ML experiments described here are evaluated with 4-fold cross-validation, which makes better use of the available corpus. The first experiment used 0.1 as the classification probability threshold, first over 229 abstracts (one per service) and then over 1019 documents (including the abstracts). The first part of Table 5 shows the traditional and BDM measures for document classification.

---

8 We used the standard ANNIE tokenizer for natural language on the HTML documents and a source code tokenizer on the abstracts. The second one splits camel-cased strings (e.g., `getUnsplicedSequence` → `get Unspliced Sequence`) as well as tokens separated by whitespace.
Table 5. Results of the first and second sets of machine learning trials (one category per document)

<table>
<thead>
<tr>
<th>Documents</th>
<th>Threshold</th>
<th>traditional %</th>
<th>BDM %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$P$</td>
<td>$R$</td>
</tr>
<tr>
<td>Abstracts</td>
<td>0.1</td>
<td>27.2</td>
<td>29.5</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>47.8</td>
<td>47.8</td>
</tr>
<tr>
<td>Abstracts &amp; HTMLs</td>
<td>0.3</td>
<td>68.0</td>
<td>40.9</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>85.3</td>
<td>19.8</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>97.9</td>
<td>8.1</td>
</tr>
</tbody>
</table>

Table 6. Results for service classification by voting for service categories from the document classifications

<table>
<thead>
<tr>
<th>Documents</th>
<th>Threshold</th>
<th>Maximum categories</th>
<th>traditional %</th>
<th>BDM %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$P$</td>
<td>$R$</td>
<td>$F_1$</td>
</tr>
<tr>
<td>Abstracts &amp; HTMLs</td>
<td>0.1</td>
<td>1</td>
<td>54.3</td>
<td>49.8</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>2</td>
<td>35.0</td>
<td>52.6</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>1</td>
<td>58.1</td>
<td>52.8</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>2</td>
<td>49.8</td>
<td>53.5</td>
</tr>
</tbody>
</table>

We noted that the performance was much better for both abstracts and HTML documents, and decided to carry out further experiments using this combination. (Working with HTMLs only is not practical since quite a few services have no HTMLs or only a few small ones, whereas every service has an abstract with some content.) Table 5 shows the results for document classification with various values for the classification threshold (the minimum probability required for the SVM to assign a classification). One normally expects increasing the threshold to increase precision and decrease recall, but the effect above 0.3 was quite severe here, so we decided to continue with just the two low threshold values. (We consider recall important because it is beneficial to try to provide at least one approximate category for every service on the portal; otherwise users are less likely to find the service at all and then improve the categorization.)

The evaluation results presented so far in this section are scored per document, with one key and one response category for each document, the key being the single category selected from the manual annotations (as explained above). But the ML tool can assign different categories to documents related to the same service, so we now consider letting each service’s documents vote (with equal weighting) to assign one or two categories to it. Table 6 presents the traditional and BDM measures for service categories derived from this process, for thresholds of 0.1 and 0.3 and for one or two categories per service.

---

9 If an HTML document can be associated with more than one service, its categorization votes for each relevant service.
It is surprising to see the precision and $F_1$ drop when each service has two categories, but we can explain this in two ways. First, many of the services had only one manual annotation, so the second automatic one counts as a spurious classification, even if it is closely related: for example, several services were automatically labelled with *Genetics* and its direct superclass *Science*, although the manual annotators only marked them as *Genetics*—these services get a 50% score. Since the ultimate purpose of this work is to help users find services on the portal, however, we do not consider this a significant fault. Second, some services which had very few documents received only one automatic category even when two categories were allowed (because all the documents were labelled with the same category); several such services automatically received one correct category but lacked the second one—these too get a 50% score.

4 Related work

As a result of the proliferation of web services and growing interest in web service discovery, browsing, composition, and related matters in recent years, several approaches have been investigated in service classification.

Belhajjame & al. [15] presented an algorithm to automatically annotate web services with information to support service discovery and composability (in particular, to allow data-driven composition of workflows). This work achieved good results for a very detailed type of annotation (concentrating on the service and operation details) over a limited domain (bioinformatics), but does not really apply to our broad coverage.

The *Woogle* web-service search engine included a clustering system [16] based on parameter names into meaningful groups to allow similarity search. This work also achieved good results, although it would not be applicable to the Service-Finder task, which uses *a priori* categories. Woogle’s approach was also dependent on UDDI repositories, which (as explained in §1) we cannot rely on.

The ASSAM (Automated Semantic Service Annotation with Machine Learning) [17] tool loads WSDL files along with previously created ontologies and provides a GUI for the user to annotate the services, and learns from previous annotations in order to show suggestions and make manual annotation faster and easier. This system uses the authors’ work on web service annotation with machine learning [18, 19], which is related to our approach presented here. That work used Naïve Bayes and SVM classification on combinations of bags of words from UDDI descriptions, WSDL files, and services’ input and output messages to classify two sets of services according to ontologies with 11 and 26 categories, respectively. The authors obtained accuracy up to around 75%, and noted that it is beneficial to offer users a choice from a small number of predicted categories (this is similar to our point that “near miss” categories are still useful for our portal users); their best result for this purpose was to get the correct category in the top three predictions 82% of the time. It is also worth noting that they obtained better results by training and applying separate classifiers for the different types of input data.
Although our results in Table 6 are not as high as ASSAM’s best [19] our task is more difficult: our service ontology has twice as many classes and a service can belong to any number of them, although ML techniques only produce one label per instance. We currently deal with the second problem by treating each document rather than each service as an instance, and then collating the results and putting an arbitrary limit on the number of categories per service.

5 Conclusions and future work

We therefore consider the results in Table 6 to be very good for our purpose, and have taken steps (classifying documents then voting to categorize services) to deal with the weakness of ML.

A useful result of these experiments for Service-Finder has been to show that it is not worth pursuing incremental improvements in the first classification approach (using keywords and rules), since it was relatively easy to get significantly better results with machine learning—which will also make use of the portal’s collaborative tools (unlike the first approach).

Future work in this classification task will incorporate several improvements. For the machine learning itself, we will experiment with using separate classifiers for abstracts and HTML documents. (Using separate classifiers for different types of input texts significantly improved the results in [18, 19].) We will also develop an ontologically aware voting system to avoid assigning near-duplicate categories: for example, if the top three voting results for a service are Finance, Currency, and Logistics, it will eliminate Finance (a direct superclass of Currency) and return only the other two, which are on separate branches.

We will also obtain manually categorized services from our users’ annotations (through the portal’s Web 2.0 functionality) to improve the training data, and continue to experiment with categorization through machine learning according to the evaluation measures used here.

Acknowledgements

This research is partially supported by the European Union’s Seventh Framework Program project Service-Finder (FP7-215876).

References