AWSC: An approach to Web service classification based on machine learning techniques

Marco Crasso, Alejandro Zunino and Marcelo Campo

ISISTAN Research Institute. UNICEN University. Campus Universitario, Tandil (B7001BBO), Buenos Aires, Argentina.
Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)
{mcrasso, azunino, mcampo}@exa.unicen.edu.ar

Abstract

A Web service is a Web accessible software that can be published, located and invoked by using standard Web protocols. Automatically determining the category of a Web service, from several pre-defined categories, is an important problem with many applications such as service discovery, semantic annotation and service matching. This paper describes AWSC (Automatic Web Service Classification), an automatic classifier of Web service descriptions. AWSC exploits the connections between the category of a Web service and the information commonly found in standard descriptions. In addition, AWSC bridges different styles for describing services by combining text mining and machine learning techniques. Experimental evaluations show that this combination helps our classification system at improving its precision. In addition, we report an experimental comparison of AWSC with a related work.

Keywords: Web services, text classification, machine learning.

1 Introduction

The software industry has broadly adopted Service Oriented Architectures by using Web service technologies. A Web service [25] is a Web accessible software that can be published, located and invoked by using the standard Web infrastructure. Current technologies for publishing Web services, for example UDDI[1], enable providers to manually assign a category to their services from a number of predefined choices such as business, educational, finance, scientific, etc. By classifying their services, providers allow consumers to search and browse available services by category or domain, this is, by taking advantage of services meta-data and thus reducing the search space.

Assigning a proper category to a service can be a tedious and error prone task due to the large number of categories usually present in Web service registries. Commonly used taxonomies such as the United Nations Standard Products and Services Code (UNSPSC)[2] or the Standard Industrial Classification (SIC)[3] contain hundreds of categories. From a service consumer point of view the situation is similar, this is, developers wanting to use a service have to manually browse published services by category. The problem is that determining the right category where to look for is often very difficult. This, besides being time consuming, may hinder the reuse of published ser-

[1]UDDI `http://www.uddi.org/`
[2]UNSPSC `http://www.unspsc.org/`
[3]SIC `http://www.osha.gov/pls/imis/sicsearch.html`

vices.

Techniques for automatically classifying Web services, this is, determining their category, will help to take advantage of the described UDDI infrastructure effortlessly. For example, these techniques will assist publishers by suggesting several candidate categories for the services they aim to publish. On the other hand, automatic classification methods will help service consumers by allowing them to query-by-example UDDI registries. In other words, users will provide a partial Web service description, and an automatic classifier will determine the most suitable categories where to look for the needed functionality. As a result, both service providers and consumers will be able to better exploit Web service technologies. Besides, several recent academic efforts [12, 19] have shown that an accurate automatic classification system is very important for Semantic Web service [26] annotation tools.

Some approaches have been proposed for automatically or semi-automatically classifying Web services [20, 5, 19, 12, 2]. However, these approaches have shown some limitations (see Section 4). First, some of them have low accuracy. Second, some of these approaches propose to classify Web services basing on the definitions of operation arguments that belong to a particular category. The main limitation of these matching approaches is that they do not attempt to reduce the distance between different styles for defining arguments present in standard descriptions. Third, some of these methods do not exploit a Web service interface description and its associated textual documentation. Finally, some of these methods assume the existence of additional sources of information, such as semantically annotated services, which might be either more expensive [17] or more challenging [26] to produce than manually classifying Web services.

This paper describes a novel method called AWSC (Automatic Web Service Classifier) for automatically classifying Web services, this is, determining the category of a Web service, given a set of predefined categories. In this paper we show that:

- By exploiting the dependencies between the category of a Web service, its provided operations and its textual documentation, namely argument definitions and comments written by developers, we build a classification system.

- By bridging different coding conventions (e.g., Hungarian notation, Java Bean notation) and different styles for defining service arguments the classification accuracy can be improved.

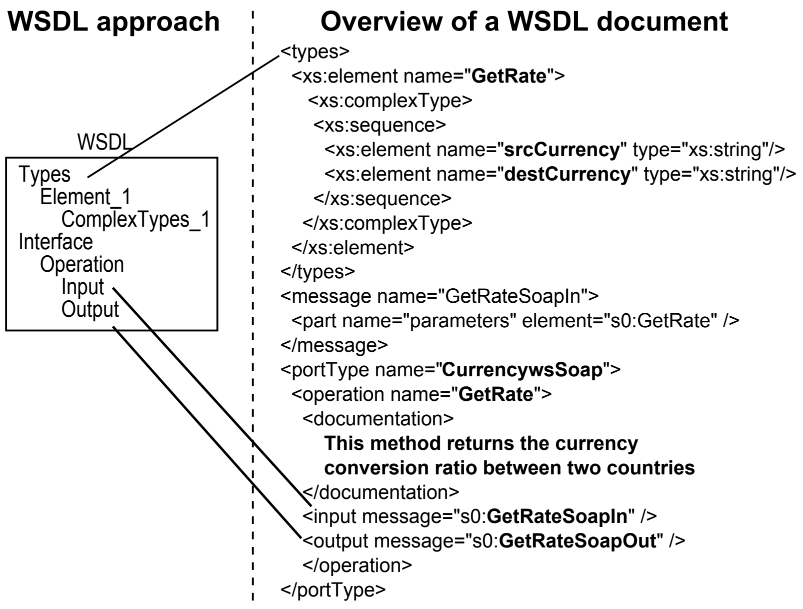
- AWSC achieves better precision than a related approach for classifying Web services. Our approach has been validated in two parts. First, by testing our classification approach on a group of 235 hand-classified Web services. Second, by comparing our Web service classification approach against a related work using a group of 391 services organized in 11 categories.

The rest of this paper is organized as follows. The next section describes AWSC, its design and its algorithms. Then, in Section 3 we report the evaluation of our approach. Section 4 discusses the most relevant previous approaches. Finally, concluding remarks and future lines of research are described in Section 5.

2 AWSC (Automatic Web Service Classifier)

The goal of AWSC is to determine the list of candidate categories of a Web service description from several predefined categories such as business, educational, finance, scientific, etc. The resulting list of candidate categories is ordered according to the confidence AWSC has about each alternative being the right one. A Web service is described by a WSDL (Web Service Description Language) document, a well structured standard for describing, in a textual manner, a Web service. WSDL is an XML format for describing a service as a set of operations, whose invocation is based on exchanging messages. In object-oriented terms, a WSDL document describes a service as an interface, an operation as a method, and a message as a method argument. The structure of messages, i.e., arguments, are defined abstractly by using the XML Schema Definition (XSD) [7]. Fig. 1 shows a WSDL document and its corresponding relevant information (in bold) for AWSC. We will use this example in the rest of the section for clarifying the main stages of AWSC.

Conceptually, we propose to classify a target Web service based on already classified similar ser-



```
<types>
  <xs:element name="GetRate">
    <xs:complexType>
     <xs:sequence>
       <xs:element name="srcCurrency" type="xs:string"/>
       <xs:element name="destCurrency" type="xs:string"/>
     </xs:sequence>
    </xs:complexType>
  </xs:element>
</types>
<message name="GetRateSoapIn">
  <part name="parameters" element="s0:GetRate" />
</message>
<portType name="CurrencywsSoap">
  <operation name="GetRate">
   <documentation>
      This method returns the currency
      conversion ratio between two countries
   </documentation>
   <input message="s0:GetRateSoapIn" />
   <output message="s0:GetRateSoapOut" />
   </operation>
</portType>
```

Figure 1: Standard Web service description

vices. To this end, AWSC compares a Web service description with other descriptions that have been manually classified. Fig. 2 depicts the overall process used by AWSC to classify a Web service. We propose a two-stage process to classify a Web service. AWSC uses text mining techniques at the first stage, namely preprocessing, to extract relevant information from a WSDL document. These techniques have been designed for preprocessing textual documentation, operations and arguments accompanying descriptions of Web services. AWSC uses a supervised document classifier at the second stage, namely classification. This classifier deduces a sequence of candidate categories for a *preprocessed* Web service description. The rest of this section will explain in detail the stages of the approach.

2.1 Text mining preprocessing for Web service descriptions

Text mining, also known as intelligent text analysis, refers to the process of extracting interesting and non-trivial information and knowledge from unstructured text [11]. In general, automatic document classifiers support classification of documents seen as objects that are characterized by features extracted from their contents, where these features may be pulled out using text mining techniques. In particular, AWSC employs text mining for extracting features from Web service descriptions. In this context, a feature is a term that is relevant to a particular category of services.

WSDL is a well structured and standard XML document format for describing Web services. Accessing a Web service description to extract category related terms is considered as a feasible task [12]. Nevertheless, Web service documentation are mostly comments written by developers, as Sabou et al. [23] assert. In general these comments are written in English, have a low grammatical quality, punctuation is often ignored and several spelling mistakes and snippets of abbreviated text are present. Also, different services are implemented by different development teams, and these teams may use different coding conventions. In addition, WSDL is a very flexible format for defining the structure of a message. WSDL allows publishers to define the same message in many styles, which may be syntactically different from each other. Broadly, a style deals with how exchanged data is encapsulated. As a result, two services conceived for a particular task may have a *syntactically* different interface, such as sendMail(ns:email e) and sendMail(xs:string from, to, subject, body).

As we will show in Section 3, these characteristics of WSDL degrade the next phase of AWSC, namely classification. Therefore, it is necessary to preprocess WSDL documents before classifying them. To address these problems we have designed some text mining preprocessing techniques that take into account the particularities of ser-

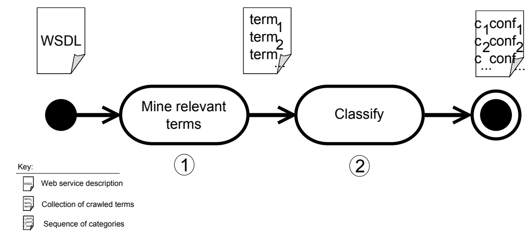


Figure 2: AWSC Web service classification process.

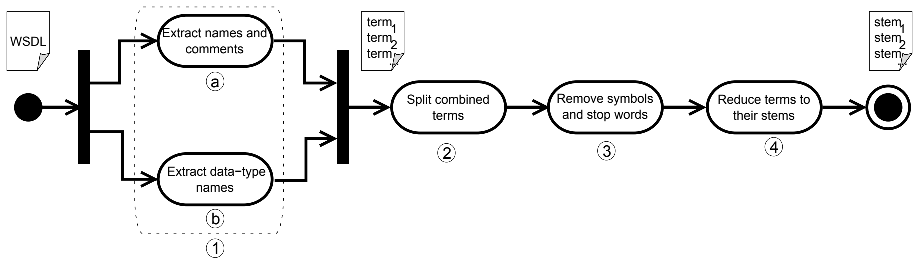


Figure 3: Text mining process.

vice descriptions. Fig. 3 depicts the overall text mining process. We have developed a parser that extracts and preprocesses the textual description within a WSDL file by taking advantage of its well structured format. We have implemented this tool in Java by using WSDL4J[4] and WSIF[5] libraries. This tool parses a WSDL document and pulls out the comments associated with the service, its operations and arguments (sub-step 1 a).

Besides, in sub-step 1 b, by executing a type enlargement algorithm our tool attempts to bridge the different encapsulation conventions mentioned. Here, if an argument data-type is non-primitive (i.e., is not an integer, string, boolean or float) the tool looks for the definition associated with this composed type and invokes a type enlargement algorithm, which extracts the names of the elements that are encapsulated in the composed type. This algorithm defines how to expand xsd:complex and xsd:element types, and then defines rules to break down other cases, e.g., arrays of "something", into these base cases. The type enlargement algorithm is:

```
procedure enlarge($type$, $follow$):string
if $follow$ is false then
  $r = type.name$
else
  if $type$ is complex then
    for $\forall s \in type.secuenceElements$ do
      $r += s.name$
    end for
  else
    if $type$ is element then
      for $\forall c \in type.children$ do
        $r +=$ enlarge( $c$, false )
      end for
    end if
  end if
end if
return $r$
```

In a second activity, by splitting combined words AWSC attempts to bridge different naming conventions. In general, developers combine a verb and a noun for denoting the name of an operation, such as getQuote or get_quote. Then, every distinct operation and message that follows each naming denomination would be treated as a different word. To bridge different naming conventions, our tool searches for combinations of words and splits them into verbs and nouns. We also consider combinations of more than two words, e.g., for "getQuoteFor" the tool dumps "get", "quote" and "for". Table 1 summarizes the rules for splitting combined words.

In a third activity, by removing symbols and stop words AWSC cleans WSDL documents. We use a list of about 600 English stop words and a list of stop words related to Web services, such as

[4]WSDL4J `http://sourceforge.net/projects/wsdl4j`
[5]WSIF `http://ws.apache.org/wsif`

Notation	Rule	Source	Result
Java Beans	Splits when changing text case.	getZipCode	get Zip Code
Hungarian	Splits when changing text case.	ulAccountNum	ul Account Num
Special symbols	Splits when either "_" or "-" occurs.	get_Quote	get Quote

Table 1: Rules for splitting combined words.

request, response, soap and post. In general, removing special characters and stop words is a simple but useful technique for filtering non-relevant terms [8]. Nevertheless, the stop words elimination process is notably improved by splitting combined words. Because of developers' coding conventions, some stop words are not removed. Besides, operations and arguments having similar features are treated as being non-similar. By splitting combined words many stop words are removed, many relevant nouns arise and many words are recognized as being common (non--relevant). Finally, we employ the Porter stemmer [21] to remove the commoner morphological and inflectional endings from words, reducing English words to their stems.

We will show the text mining preprocessing stage of AWSC with an example using the description of a Web service for finding currency exchange rates (see Fig. 1). Table 2 shows the original comment and the collection of stems generated by applying the preprocessing techniques. By using our text mining process we pulled out all relevant stems. For example, by preprocessing the message "GetRateSoapOut" we removed the stop words "get", "soap" and "out", while the relevant word "rate" arose. By expanding elements data-type definitions we included more occurrences of the relevant words in the result. For example, by expanding the data-type associated with the message "GetRateSoapIn" we derived the stems "src", "currenc", "dest" and "currenc" from the combined words "srcCurrency" and "destCurrency". In Section 3, we will evaluate how this text mining process improves the overall accuracy of our classification approach.

2.2 Web services classification

Document classification refers to the process of assigning an electronic document to one or more categories based on its contents [24]. Automatic document classifiers support classification of documents seen as objects characterized by features extracted from their contents. In general, when some external mechanism, such as human feedback, provides information on the correct classification for documents, we talk about supervised document classification. This approach consists of two phases: (1) training phase, and (2) classification phase. During the training phase, such a learning system receives a collection of categorized documents and builds a classifier. Then, during the classification phase, this classifier deduces one or more categories for a new document. The cornerstone behind AWSC supervised classification approach is the fact that there are dependencies between the category of a Web service and its description. As in a related approach [12], AWSC also considers the dependencies between the category of a Web service, the operations and their input and output arguments. Concretely, AWSC assumes that:

1. The category of a Web Service depends on its textual comments

2. and its method signatures.

The hypothesis about the dependency between categories and Web service documentation has been evaluated in related approaches [12, 23]. Conversely, according to the mentioned different styles used for defining the interface of a Web service operation, the second part of this hypothesis is not obvious unless we attempt to bridge different encapsulation and naming approaches. It is desirable to measure the dependencies between categories and arguments, i.e., operation interfaces. To this end, we conducted several experiments with variables representing both arguments and categories of services for assessing the degree of dependency between the category and the signature of an operation (see Section 3). These experiments showed that the degree of dependency increases proportionally to the number of times an argument appears in the services of this particular category, but this dependency is offset by how common the argument is in the whole collection, for the data-set used in the experiments.

Table 2: Example of results obtained by the preprocessing technique.

Extracted words	Preprocessed stems
CurrencywsSoap GetRate This method returns the currency conversion ratio between two countries GetRateSoapIn GetRateSoapOut GetRate srcCurrency destCurrency	currenc rate method currenc convers ratio countr rate rate rate src currenc dest currenc

As a result of the previous experiments, we propose a supervised classifier based on Rocchio's algorithm with TF-IDF. Rocchio is a learning algorithm, originally designed to use relevance feedback in querying full-text databases [22], which has been adapted to text classification [14]. This algorithm has a configurable word weighting method. TF-IDF is a word weighting heuristic that determines that a word is important for a document if it occurs often on it. Instead, words that occur in many documents are rated as less important because of their low inverse document frequency. Formally: for each term t_i of a document d, $tfidf_i = tf_i \bullet idf_i$, with:

$$tf_i = \frac{n_i}{\sum_{j=1}^{T_d} n_j} \tag{1}$$

where the numerator (n_i) is the number of occurrences within d of the term being considered, and the denominator is the number of occurrences of all terms within d (T_d), and:

$$idf_i = \log \frac{|D|}{|\{d : t_i \,\epsilon\, d\}|} \tag{2}$$

where $|D|$ is the total number of documents in the corpus and $|\{d : t_i \,\epsilon\, d\}|$ is the number of documents where the term t_i appears.

The overall phases of the classification algorithm are described subsequently. Initially, the algorithm represents every document as a vector $\vec{v} = (e_0, ..., e_n)$. Each element e_i represents the importance of a distinct word w_i for that document. This importance is calculated according to the selected word weighting method, TF-IDF in this case. Documents with similar content have similar vectors. Then, during the training phase, each category C_i is represented as a vector $\vec{c_i}$. This vector stands for the documents that belong to category C_i. Formally:

$$\vec{c_i} = \alpha \frac{\sum_{\vec{d} \epsilon C_i} \vec{d}}{|C_i|} - \beta \frac{\sum_{\vec{d} \epsilon D - C_i} \vec{d}}{|D - C_i|}$$

with C_i being the sub-set of the documents from category i, and D the amount of documents of the entire data-set. First, both the normalized vectors of C_i, i.e. the positive examples for a class, as well as those of $D - C_i$, i.e. the negative examples for a class, are summed up. The centroid vector is then calculated as a weighted difference of the positive and the negative examples. The parameters α and β adjust the relative impact of positive and negative training examples. According to [14], we use $\alpha = 16$ and $\beta = 4$. Additionally, Rocchio requires that negative elements of the vector are set to zero. During the classification phase, a new document is represented as a vector and then compared to the vectors associated with all categories by using cosine similarity [24]. Finally, the category that maximizes vector similarity is deduced.

By treating each preprocessed WSDL as a document we have developed a Rocchio classifier for Web services. In this way, we first employ the preprocessing method described in Section 2.1 for pulling out all stems contained in each WSDL file. Then, the resulting collection of stems stands for a document associated with a Web service. Second, for representing these documents in a vector space and building the classifier, AWSC uses a software that performs statistical text classification named Rainbow [16].

In Section 3, we show how the accuracy of this classifier surpasses that of two well-known classifiers and discuss a related approach.

3 Evaluation

This section describes the experimental evaluation of AWSC. First, we measured the dependencies between categories and method signatures. We have assessed that an argument is more significant to a category, if it has a high importance in the given category but a low importance in the whole collection of categories. Here, "importance" refers to the TF-IDF value for an argument. We have verified our hypothesis on a subset of the Web services collection presented in [12]. This subset is composed of 235 hand-classified Web services, as shown in Table 3. This collection was obtained from SALCentral and XMethods repositories. It contains a plain text description for each service and the WSDL document associated with each service.

Then, we pulled out argument declarations from each WSDL document. As a result, complex type arguments were expanded and their names were reduced into stems. Afterwards, we worked out a comma separated file whose first column represents the category associated with the Web service, and the second one represents input arguments. We tried to find out interface patterns within category related services by using Association Rules, specifically, the Tertius [9] algorithm from Weka. Some discovered rules are presented below. The number on the right of each rule is a measure of how much evidence exists for that rule (a.k.a. confirmation value):

1. input = send → category = c_2 (0.1531)
2. input = zip → category = c_0 (0.1001)
3. input = quot → category = c_1 (0.0818)
4. input = chang → category = c_3 (0.0788)
5. input = messag → category = c_2 (0.1069)

Although the results from our first experiment have shown that there were some degrees of dependency between the category of a service and its inputs, the discovered rules were far away from becoming the bases of a classification system, mainly due to the variety of argument names that Web services can employ. We conducted a second experiment, with the same data-set, to measure the degree of dependency between the interface of a Web service and the category this service belongs to. We normalized the occurrences of an argument within a category as shown in Table 4.

The second experiment has shown that the importance of an argument to a particular category increases proportionally to the number of times an argument appears in the services of this particular category, but this importance is offset by how common the argument is in the whole collection. For example, the stem "quot" was associated with the category "financial", whereas the stem "search" was connected to all categories. Then, an operation with an argument named "quot" had a good chance of being part of a service that belonged to the category "financial". Conversely, if an operation had an argument named "search", we were not able to deduce its category accurately. This result is coherent with the word weighting method used by AWSC, namely TF-IDF.

Second, we have shown that by using Rocchio with TF-IDF, AWSC achieves better results than using K-NN [4] and Naïve Bayes [15] (see Table 5), using the aforementioned data-set. Despite K-NN might have a limited ability to deal with data-sets with unrepresentative features, we think that it was worth testing its performance because of its simplicity and efficiency. To compare the Rocchio classifier with the two just mentioned, we drove a repeated holdout experiment. In the simplest form of a holdout experiment, observations are chosen randomly from the initial sample to form the validation data and the remaining observations are retained as the training data. In the repeated mode, results from successive tests with different training sets, but of the same size, are mediated. Then, we used the same corpus and destined the same number of instances to the training and classification phases (90% of the entire collection destined to the training phase), to run successive tests with each classifier. We evaluated one thousand random distributions for every classifier by using the Rainbow toolkit [16] and Weka [10].

Third, we measured the accuracy of AWSC with different training set sizes. We have shown that AWSC accuracy averaged 85%, when at least 210 Web services were randomly chosen for training (less than 50 services per category). Besides, the average accuracy improved near-linearly as we increased the number of services destined to the training phase. It is worth noting that with a small training set (60% of the collection), the accuracy averaged almost 81%. To evaluate how many training samples were required to begin deducing accurately, we drove four repeated holdout experiments. We evaluated four random distributions (60%, 70%, 80% and 90% of the entire collection destined to the training phase). For each distribution we executed one thousand rounds, and calculated an average again. The average accuracy is shown in Fig. 4.

Fourth, we compared the accuracy obtained by using type expansion and verb noun separation versus not using them. The achieved results have shown that by applying our proposed text min-

Table 3: Web services collection.

Service category	Samples	Unique words
Country Information (c_0: countryInfo)	50	1404
Financial (c_1: financial)	49	2954
Communication (c_2: communication)	41	791
Unit Conversion (c_3 : converter)	49	1067
Search Engine (c_4: search)	46	2135

Table 4: Importance of the stems within a particular category.

Stem/Category	c_0	c_1	c_2	c_3	c_4
zip	0.329	0.062	0	0	0
quot	0	0.434	0	0	0
messag	0	0.031	0.443	0	0.022
chang	0	0.031	0	0.559	0
search	0.019	0.031	0.109	0.014	0.253
code	0.307	0.238	0.093	0	0
quer	0	0.041	0	0	0.159

Table 5: Comparison between different classifiers.

Classifier	Average Accuracy
K-NN	39.59%
Naïve Bayes	79.38%
Rocchio	85.08%

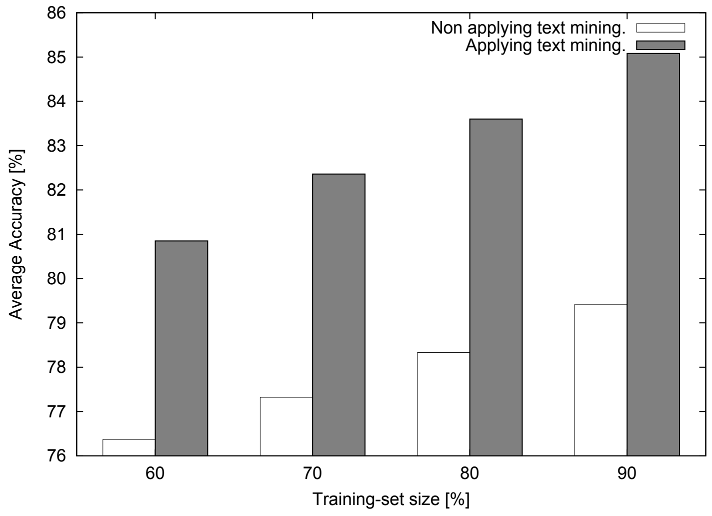


Figure 4: AWSC classification accuracy varying the training set size (with/without text mining).

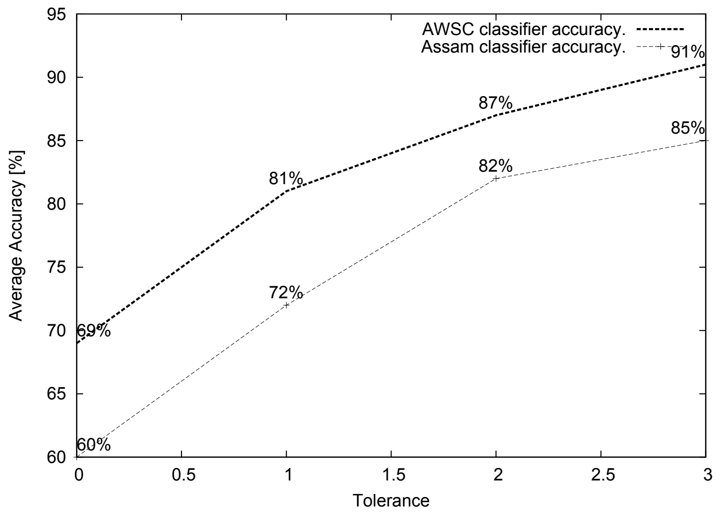


Figure 5: Comparison between Assam and AWSC.

ing process over this data-set improves the overall accuracy of the Rocchio classifier in, at least, 5%. Then, we conducted another repeated hold-out experiment. We first adapted our text mining process for omitting the techniques just mentioned. Notably, the resulting extracted text differed from the original results in having more unique words per category. For example, there were 2954 unique words in the "financial" category after preprocessing, and 7548 words before. This was caused by the inclusion of many irrelevant words, as explained in Section 2.1. The inclusion of such irrelevant words does not contribute to the classifier. On the contrary, it is expected that such words could harm the classifier performance [8]. At this point, we had two collections of documents, the collection of preprocessed Web service descriptions and the collection of Web service descriptions per se. Therefore, we trained two Rocchio TF-IDF classifiers for running successive tests with each one. It is worth noting that we also compared how these classifiers worked when varying the training set size. Fig. 4 shows the results.

Finally, we compared AWSC to a related proposal named Assam [13]. In order to compare them, we reproduced the experiment reported in [13] with the same data-set and evaluation methodology, but using our classifier. It is worth noting that Hess et al. only reported the accuracy of their classification approach. Assam was evaluated with a group of 391 Web services divided in 11 categories, which has been made publicly available. Its accuracy was evaluated with a tolerance value and by driving a leave-one-out experiment. Leave-one-out is a cross-validation method for evaluating classifiers [27]. A tolerance value of t represents that the correct classification is included in a sequence of $t+1$ suggestions. Basically, in a leave-one-out experiment one observation is arbitrary chosen from the initial sample to form the validation data and the remaining observations are retained as the training data. Therefore, we preprocessed the 391 Web services. Afterwards we built the classifier and drove a leave-one-out experiment. AWSC accuracy surpassed Assam accuracy in, at least, 9% with two values of tolerance: $t = 0$ and $t = 1$. The results are shown in Fig. 5.

To sum up, we have shown that Rocchio with TF-IDF is a proper classification technique for Web services. Besides, our text mining techniques have improved the classifier accuracy in, at least, 5%, resulting in an average accuracy of 85%. It is worth noting that this accuracy has been achieved with less than 50 services per category destined for training. In order to conduct a more formal comparison of these classifiers it would be necessary to assess the standard deviation statistic of Assam and AWSC, however to the best of our knowledge the authors did not report it in [13] or [12]. Finally, we have shown that AWSC achieved more accuracy than others classifiers and Assam. Although we have evaluated AWSC with a well known set of Web services, the reported results may vary with different data-sets.

4 Related Work

During the past few years some efforts and research have been placed on assisting the developer to classify Web services. As a result, some semi-automatic and automatic methods have been proposed. The different approaches are based on argument definitions matching [20, 5], document classification techniques [19, 13] and semantic annotations matching [2]. In the rest of this section we review and discuss these approaches.

MWSAF [20] is an approach for classifying Web services based on argument definitions matching. First, MWSAF translates these definitions into a graph. MWSAF also translates real word formal descriptions of categories (a.k.a. ontologies) that have been specified by using Semantic Web languages. Then, MWSAF uses graph similarity techniques for comparing both. Likewise, Duo et al. [5] translate a definition into an ontology, instead of into a graph. Then, an ontology alignment technique attempts to map one ontology onto another [6]. The main limitation of these matching approaches is that they do not attempt to reduce the distance between different coding conventions. In fact, MWSAF achieves low accuracy, which is shown in its experimental evaluations.

METEOR-S [19] describes a further improved version of MWSAF. The problem of determining a Web service category is abstracted to a document classification problem. The graph matching technique is replaced with a Naïve Bayes classifier. To do this, METEOR-S extracts the names of all operations and arguments declared in WSDL documents of pre-categorized Web services. The main limitation of this approach is that it assumes independence between the name of an operation and its arguments. Clearly, the name, or header, of an operation and the name of its arguments might be related, e.g., print(Style, Document) method signature. Therefore, this classifier seems to be based on a false premise. Although METEOR-S proposes a document classification approach, natural language documentation, usually present in WSDL files and service registries, is not considered. However, METEOR-S experimental results, for the same small data-set previously mentioned, shows an accuracy improvement with respect to MWSAF.

Assam [12] is an ensemble machine learning approach for determining Web service category. Assam combines the Naïve Bayes and SVM [3] machine learning algorithms to classify WSDL files in manually defined hierarchies. Assam takes into account Web service natural language documentation and descriptions. As reported, this approach is more accurate than similar approaches, even though its authors used a repository of 391 Web services divided into 11 categories for experimenting, which makes the validation of Assam stronger than the others. This repository[6] has been made publicly available, which by itself has been an important contribution to the field. The evaluation shows that when suggesting a domain for annotating a Web service the accuracy was 60% for a tolerance value of $t = 0$, and it trended towards 90% for $t = 10$.

Previous efforts for classifying Web services have several shortcomings. First, the classification approach proposed by MWSAF has shown low accuracy. Even though this work was evaluated with a set of *only* 24 services divided in two categories, the resulting accuracy was 0.625. Second, the *classification-based* version of MWSAF shown better accuracy, but this version is based on the false premise that an operation and its argument names are independent. In addition, this approach does not consider natural language documentation. Third, Assam uses an ensemble machine learning technique that suffers from low accuracy when $t = 0$ (automatic mode). Fourth, the computational complexity of the techniques described in [20] and [5] is directly proportional to the number of entities (classes, properties, etc.). Therefore, both approaches might have problems handling a large number of ontologies. Finally, the described proposals classify new services based on the information present in their descriptions. Conversely, [2] use semantically annotated services as additional sources of information. The problem of this work is that there are too few annotated services, because of the important effort required to semantically annotate Web services [17]. Therefore, this semantic approach proposed suffers from a cold-start problem because it assumes that a corpus of previously annotated services is available.

5 Conclusions

Despite the important benefits Web services provide, Web service technologies are not as broadly

[6]Categorized Web services Repository, http://msi.ucd.ie/RSWS

shared and reused across application, enterprise, and community boundaries as one may expect. One of the factors that hinders the adoption of Web Services is that manually assigning a proper category for a Web service description is very difficult. We aim to provide an approach for easing this. We tackle the complex task of automatically classifying services by exploiting standard WSDL descriptions.

Text mining and machine learning have shown to be suitable techniques for automatic classification and labeling of documents. We propose AWSC, a novel method that combines text mining and machine learning techniques for classifying Web services. As reported, Rocchio with TF-IDF is a proper classification technique for Web services. Besides, by combining the text mining and classification stages of AWSC the resulting accuracy was improved in, at least, 5%, using a well known data-set. We have shown that the average accuracy of AWSC Web services classifier was 85%, even with a few services destined for training. In addition, we have shown that AWSC accuracy surpassed a related work based on Naïve Bayes and SVM in, at least, 9%.

Although we have evaluated AWSC with a well known set of Web services, the reported results may vary with different data-sets. It would be interesting to evaluate AWSC with more data--sets. However, as far as we know no other data-set or benchmarks have been made publicly available yet.

The main limitation of AWSC is that it assumes that a corpus of previously classified services is available. This generates the inability for dynamically creating categories without re-building the classifier. To cope with such a requirement, an incremental clustering [1] approach might be more suitable than a classification one.

We are analyzing clustering approaches to evaluate how these methods impact on the overall performance of AWSC. In addition, we are planning to incorporate semantic annotating support to AWSC. The idea is to transparently associate Web services to semantic meta-data based on similar services that have been previously annotated. In this context, a sensible classification system as AWSC may "guide" the annotating process by deducing a handful set of similar services.

Acknowledgements

This work is partially supported by the National Council of Technical and Scientific Research of Argentina (CONICET). We thank the anonymous referees whose insightful comments and recommendations have helped to improve the quality of this paper.

References

[1] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley, 1999.

[2] Miguel Ángel Corella and Pablo Castells. Semi-automatic semantic-based Web service classification. In *Business Process Management Workshops*, volume 4103 of *LNCS*, pages 459–470, Vienna, Austria, September 4-7 2006. Springer.

[3] Nello Cristianini and John Shawe-Taylor. *An introduction to support Vector Machines and other kernel-based learning methods.* Cambridge University Press, New York, NY, USA, 2000.

[4] Belur V. Dasarathy. *Nearest neighbor (NN) norms: NN pattern classification techniques.* IEEE Computer Society Press, 1990.

[5] Zhang Duo, Li Zi, , and Xu Bin. Web service annotation using ontology mapping. In *IEEE International Workshop on Service-Oriented System Engineering*, pages 235–242, 2005.

[6] Marc Ehrig and Steffen Staab. QOM - quick ontology mapping. In McIlraith et al. [18].

[7] David C. Fallside and Priscilla Walmsley. XML Schema part 0: Primer second edition. W3C recommendation, World Wide Web Consortium, October 2004. `http://www.w3.org/TR/xmlschema-0/`.

[8] Ronen Feldman and James Sanger. *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data.* Cambridge University Press, 2006.

[9] Peter A. Flach and Nicolas Lachiche. Confirmation-guided discovery of first-order rules with Tertius. *Machine Learning*, 42(1/2):61–95, January 2001.

[10] Eibe Frank, Mark A. Hall, Geoffrey Holmes, Richard Kirkby, and Bernhard Pfahringer. WEKA - A machine learning workbench for data mining. In *The Data Mining and Knowledge Discovery Handbook*, pages 1305–1314. Springer, 2005.

[11] Marti A. Hearst. Untangling text data mining. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 3–10, Morristown, NJ, USA, 1999. Association for Computational Linguistics.

[12] Andreas Heß, Eddie Johnston, and Nicholas Kushmerick. ASSAM: A tool for semi-automatically annotating semantic Web services. In McIlraith et al. [18], pages 320–334.

[13] Andreas Heß and Nicholas Kushmerick. Learning to attach semantic metadata to Web services. In Dieter Fensel, Katia P. Sycara, and John Mylopoulos, editors, *International Semantic Web Conference*, volume 2870 of *Lecture Notes in Computer Science*, pages 258–273. Springer, 2003.

[14] Thorsten Joachims. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In *International Conference on Machine Learning*, pages 143–151, Nashville, TN, USA, July 8-12 1997. Morgan Kaufmann.

[15] David D. Lewis. Naive (Bayes) at forty: The independence assumption in information retrieval. In *10th European Conference on Machine Learning*, volume 1398 of *LNCS*, pages 4–15, Chemnitz, Germany, April 21-23 1998. Springer.

[16] Andrew Kachites McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. `http://www.cs.cmu.edu/~mccallum/bow`, 1996.

[17] Rob McCool. Rethinking the semantic Web. Part I. *IEEE Internet Computing*, 9(6):86–88, 2005.

[18] Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors. 3^{rd} *International Semantic Web Conference*, volume 3298 of *LNCS*, Hiroshima, Japan, November 7-11 2004. Springer.

[19] Nicole Oldham, Christopher Thomas, Amit P. Sheth, and Kunal Verma. METEOR-S Web service annotation framework with machine learning classification. In *Semantic Web Services and Web Process Composition*, volume 3387 of *LNCS*, pages 137–146, San Diego, CA, USA, 2004. Springer.

[20] Abhijit A. Patil, Swapna A. Oundhakar, Amit P. Sheth, and Kunal Verma. METEOR-S Web service annotation framework. In *Proc. of the* 13^{th} *international conference on WWW*. ACM Press, 2004.

[21] Martin Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, July 1980.

[22] Joseph J. Rocchio. Relevance feedback in information retrieval. In *The Smart retrieval system - experiments in automatic document processing*, pages 313–323, Englewood Cliffs, NJ, USA, 1971. Prentice-Hall.

[23] Marta Sabou, Chris Wroe, Carole A. Goble, and Heiner Stuckenschmidt. Learning domain ontologies for semantic Web service descriptions. *Journal of Web Semantics*, 3(4):340–365, 2005.

[24] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002.

[25] Steven J. Vaughan-Nichols. Web services: Beyond the hype. *Computer*, 35(2):18–21, February 2002.

[26] Kunal Verma and Amit Sheth. Semantically annotating a Web service. *IEEE Internet Computing*, 11(2):83–85, 2007.

[27] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques.* Series in Data Management Systems. Morgan Kaufmann, 2^{nd} edition, June 2005.