ISSN 1870-4069

WSDL Information Selection for Improving Web Service Classification

Christian Sánchez-Sánchez[1], Esaú Villatoro-Tello[1],
Gabriela Ramírez-de-la-Rosa[1], Héctor Jiménez-Salazar[1], David Pinto[2]

[1] Universidad Autónoma Metropolitana Unidad Cuajimalpa,
Departamento de Tecnologías de la Información,
División de Ciencias de la Comunicación y Diseño, México

[2] Benemérita Universidad Autónoma de Puebla,
Facultad de Ciencias de la Computación, Puebla, México

{csanchez, evillatoro, gramirez, hjimenez}@correo.cua.uam.mx,
dpinto@cs.buap.mx

Abstract. Currently, the increasing number of available Web Services (WS) over the Internet has induced the urgency for proposing new ways for searching and categorizing such software pieces. Normally, WS functionality is detailed through the WSDL description language, resulting in a structured document that includes a great variety of features definition. One of the WSDL inner features *"documentation"* is designed to describe the Web Service functionality, in natural language, which could help to classify and find WS. Nevertheless, the majority of WS lack of that description. To tackle this problem, this paper presents an analysis of the WSDL inner feature information that can assist to classify WS, without any extra data. The experiments carried out on three different WSDL collections showed that only with minimal information is possible to increase the performance of automatic WS classification.

Keywords. Web service classification, WSDL information analysis, feature selection.

1 Introduction

Nowadays, in the field of Software Development exists a strong motivation for preserving and encouraging the use of certain programming styles and conventions as recommended practices. Among the advantages of such conventions are the following: *i)* high efficiency of software applications by means of distributed systems, *ii)* collaborative applications through standard mechanisms, *iii)* loosely coupled systems that allow software reuse, and, *iv)* costs reduction during the software development phase.

Accordingly, Web Services (WS) emerged as software development mechanisms that allow developers to fulfill previously mentioned characteristics. A WS can be thought as a web application which uses XML based standards for

communicating with external systems for providing the necessary service for the user [2]. Web services being a business trend over software applications, contain encapsulated descriptions of their functionality (*i.e.*, methods and functions, usually known as *description features*) described as an abstract interface by means of using standard Web Services Description Language (WSDL). For locating desired services, users appeal for public registries like UDDI (Universal Description, Discovery and Integration) where by means of matching their requirements to a set of registered services, users are able to obtain "relevant" services. However, search functionality still is very simple and fails to account for relationships between WS and users' real needs [3]. A bigger problem is that not all WS developers register their services and with the ever-increasing amount of published WS on the Internet, the task of finding the correct service has become a challenging issue in service-oriented computing [2, 3].

Another important aspect to consider is that in spite of fact that the WSDL description is a structured document, it is hard for a common Internet user to understand its content. One of the WSDL inner features *"documentation"* is designed to describe the Web Service functionality in natural language, which could help to classify and find WS. Nevertheless, it is very common that suppliers do not include such *documentation*.

This is why the necessity of creating algorithms or methods that help in the process of categorize and search WS becomes important. For this reason, in order to fulfill that necessity, some questions were stated. *(a)* Is it possible to extract enough understandable information (words) from WSDL documents?, and *(b)* which information, from WSDL inner features, improve WS classification?

In order to answer these questions, this paper presents an analysis of the WSDL inner feature information that can assist to classify WS without any extra data. One characteristic of such information is that is expressed by words.

For the experiments we used WSDL standard collections, namely OWLS-TC[3] v3 and v4, and ASSAM[4]. We show that some of the evaluated features improve the classification performance achieving competitive results to the state of the art.

The rest of this document is organized as follows. Section 2 presents some related work concerning the automatic web services classification task. Section 3 describes how an WSDL document is conformed. Then, Section 4 describes used datasets, experimental setup and obtained results by our proposed approach. Finally, Section 5 depicts our conclusions and some future work directions.

2 Related Work

In the literature of WS classification, two different types of approaches can be distinguished. Those that use external resources to build the classification method, and those that do not need any external resources but just the *description* features contained in the WSDL document.

[3] http://projects.semwebcentral.org/projects/owls-tc/
[4] http://www.andreas-hess.info/projects/annotator/

On the one hand, among the external resources used by some approaches, the most widely used is The United Nations Standard Products and Service Code (UNSPSC), a standard taxonomy used to manually classify WS. This taxonomy defines a five-level and tree-structured hierarchical classification. The UNSPSC taxonomy was used by [9] to automatically classify WS using clustering algorithms in two steps. First, it considers the terms contained in the metadata of the WSDL documents to generate a tree structure representation of the WS, and then it considers the underlying semantic relation among metadata structures, such as, terms co-occurrences of words taken from the input, output and function descriptions of the WSDL document: *its description features*. After these two clustering results, Liang et al. [9] uses the taxonomy to assign a class to a WS tree.

In the approach proposed by Wang et al. [16] the UNSPSC taxonomy is used to generate a set of vectors for the training phase of the SVM algorithm. Given a set of domains, each subtree found in the taxonomy under these domains are treated as concept of that domain. They claim that the functional description of a WS is always related to a set of concepts. Therefore, a vector is constructed for each concept to represent a training document for the SVM algorithm. In addition to UNSPSC taxonomy, in this work they used WordNet to provide semantic similarity of concepts to weight the terms in the vector space model. WordNet as external resource was also used by Boujarwah et al. [4] as a lexical English database to generate conceptual graphs for each domain. Then, they used the conceptual graphs generated to classify a new WS.

Another example of this kind of approaches is introduced by Yang et al. [18]. Using OWLS-TC4 dataset, words are extracted from the WSDL. An external resource is used in order to identify abbreviations or if they are nouns or verbs. The pre-processing step involves: splitting (verbs and nouns), eliminating stop words, stemming and removing specific tags (web, service, input, output). Then they tested 4 different classification algorithms Support Vector Machines (SVM), Naive Bayes (NB), Decision Tree (DT) C4.5 and Neuronal Networks. Only words from names of services, operations, inputs and outputs were extracted. The best results for classification was obtained using output name words and applying C4.5 DT Algorithm.

In the approach introduced by Nisa et al. [11] were extracted: service name, service documentation, WSDL messages, WSDL ports and WSDL schema from WSDL documents, in order to classify web services using text mining. For each extracted feature that was classified, using Maximum Entropy, a comparison of the accuracy, through different categories, was done. The best results were obtained including WSDL Schemas information. In this paper it is also showed a comparison of the effects of using some preprocessing like: steeming, lemmatization and word splitter. The results were improved using the last two. Unfortunately, their dataset is not available for comparison.

On the other hand, there are WS classification methods developed without using any external resource to classify WS, these methods usually rely on the WSLD *description* features only (Section 3). For instance, Saha et al. [12]

propose a WS representation based on Tensor Space Model (TSM) in order to capture the internal structure of WSDL documents. The method consists in selecting a set of relevant tags from a WSDL document. For each tag, they build a tensor using all words under that particular tag. For each tensor they apply a classification algorithm that gave independent classification results and as a final step they combine all this information using rough sets. Another example is given by Bruno et al. [5], where for classifying a WS, authors identified key concepts in the WSDL *description features*, and then by means of a SVM algorithm, they classify each WS into a specific domain.

Notice that all related work shown above used, in some way, the information of the WSDL description file. However, some works use it merely as source of terms to construct more elaborate representation, and some of them use it as the main source of information to the classification. Consequently, these previous research demonstrate, to some extent, that there exists a relationship between WS functionality and the information contained in the WSDL document, opposite to what Lu *et al.* claim [10].

Contrary to previous work, our proposed approach do not depend on any external resources during the classification phase. Also, the information gotten from the WSDL inner features are words, many of previous approaches worked using strings.

3 A WSDL Document

As we have mentioned in previous sections, employing the WSDL language for describing WS functionality is a recommended practice among software developers. Accordingly, as examples of the information that a user can obtain from reviewing the WSDL document (*i.e., the description features*) we have the following:

- **Types**- a container for data type definitions using some type system (*i.e., data types*).
- **Messages**- an abstract typed definition of the data being communicated, messages normally tend to include parameters information and communication protocols.
- **Operations**- an abstract description of all actions supported by the web service (*i.e.,* methods' *names*).
- **Documentation**- natural language description of the full functionality (operations) of the web service (usually missing).
- **Port Type**- an abstract set of operations supported by one or more endpoints.
- **Binding**- a concrete protocol and data format specification for a particular port type.
- **Port**- a single endpoint defined as a combination of a binding and a network address.
- **Service**- a collection of related endpoints.

Fig. 1. An excerpt of a real WSDL document from the domain FOOD extracted from the OWLS-TC V3 collection. Notice that any *documentation* has been provided.

According to Stroulia et al. [15], information like services' names associated to its methods, parameters and data types are useful since they reflect (to some extent) the semantics of the underlying capabilities. For this reason in the proposed experiments, it is contemplated extracting information from inner features and test which of them are useful to improve classification. Next, more about the information extraction algorithm and document representation is given.

3.1 Information Extraction Algorithm

For performed experiments we considered the following features: *Service Name*, *Operations*, *Documentation*, *Messages*, and *Types*, configured in different ways. Observe that only the *Documentation* feature represents a text described in natural language, hence, extracting information of such feature do not represent a complicate process. However, for the rest of the considered features, extracted terms do not represent well formed words (See Figure 1), therefore we followed the steps described in Algorithm 1 in order to extract more readable information (words).

Notice that the proposed algorithm uses a method named **generateSubstrings**, which aims at extracting all possible sub-strings from any composed expression e_i extracted from the *description features*. Proposed method performs the following steps:

- Obtains all possible sub-strings from expression e_i by means of computing all the combinations of consecutive characters from e_i.
- Obtains all sub-strings that start with a capital letter. Intuitively, composed expressions will contain in fact several words, and as a common programming practice, developers tend to set the first letter of each word in its capital form.
- Obtains all possible sub-strings from e_i separated by some special character. Similarly to previous point, it is also a common practice among developers to use some set of special characters to separate composed expressions (*e.g.*, _ – =:,).

Finally, as can be noticed in Algorithm 1, once all possible sub-strings were obtained, each word is checked for its existence within WordNet. We verify if

that word is not a stop word[5]. Thus, at the end we have a set of words, extracted from the *description features*. And those extracted words are used for classifying the WS. It is worth mentioning that our proposed method employs WordNet for identifying readable words only, but not for the classification process nor to include other terms, which has been the traditional approach from previous works (see section 2).

```
   input : expression e_i extracted from the description
           features and a list of stop words swList
   output: list (listOfWords) of readable words extracted
           from e_i
 1 //If the given expression e_i is a known word in WordNet (WN)
    then listOfWords is composed by e_i
 2 if ((e_i ∈ WN) and (e_i ∉ swList)) then
 3 |   Add e_i to listOfWords;
 4 |   return listOfWords;
 5 end
 6 //If FALSE, we begin the process to extract all possible
    words from e_i
 7 else
 8 |   substrings ← generateSubstrings(e_i);
 9 |   //For each obtained sub-string from e_i we verify if it
   |    represents a readable word
10 |   for i ← 1 to |substrings| do
11 |   |   if ((substrings[i] ∈ WN) and (substrings[i] ∉ swList)) then
12 |   |   |   Add substrings[i] to listOfWords;
13 |   |   end
14 |   end
15 |   return listOfWords;
16 end
17 Exit;
```

Algorithm 1: Proposed algorithm for extracting information from WSDL features.

3.2 Document Representation

As we have mentioned before, we face the problem of Web Services classification as a *document classification* task. Although document classification subsumes two types of text analyses: clustering and categorization. The difference between

[5] Normally, stop words are formed by short function words, such as *the, is, at, which*, and *on*; *i.e.,* prepositions, pronouns, etc.

the two is that the latter uses a predefined number of classes or categories with their corresponding tags, whereas in the former approach, the number and the tag for each category is to be discovered. Since in categorization the classes are known a priori, categorization algorithms usually take advantage of them by using supervised algorithms with some kind of training step.

Accordingly, Text Categorization (TC) is the task of automatically sorting a set of documents into categories (or classes, or topics) from a predefined set [13]. In its simplest form, the text classification problem can be formulated as follows: Given a set of training documents $\mathcal{D}_{Tr} = \{d_1, \ldots, d_n\}$ and a set of predefined categories $\mathcal{C} = \{c_1, \ldots, c_m\}$, the goal of TC is to devise a learning algorithm that is able to generate a classification model (*i.e.,* hypothesis) $h : \mathcal{D} \rightarrow \mathcal{C}$ that will be able to accurately classify unseen documents from $\mathcal{D}$.

The design of learning algorithms for text categorization has usually followed the classical approach of the pattern recognition field, where data instances (*i.e.,* documents) first undergo an appropriate representation. Accordingly, the first step corresponds to the *indexing* of training documents ($\mathcal{D}_{Tr}$), where each document d_j is transformed into a compact form of its content. Commonly, each document is represented as a vector of weighted terms; such idea is taken from the Vector Space Model proposed in the field of Information Retrieval [1]. Therefore, given a document $d_j \in \mathcal{D}_{Tr}$, it is represented as a vector $\overrightarrow{d}_j = \langle w_{kj}, \ldots, w_{|\tau|j} \rangle$, where τ depicts the *dictionary*, *i.e.,* the set of different terms (words) that appear at least once in some document of $\mathcal{D}_{Tr}$, and w_{kj} establishes the importance of term t_k within document d_j.

Normally, τ is obtained from filtering words from the document collection, in other words, τ is the result of a pre-processing step. As pre-processing step all stop-words were removed. Once τ has been defined, in order to represent the documents $d_j \in \mathcal{D}_{Tr}$, we employed the well known Bag of Words (BOW) paradigm.

The BOW representation has been the traditional form for representing documents [13]. Such approach employs single words as elements of the vector of terms. There are several proposals for computing the weight $w_{k,j}$ of each term (*i.e.,* the importance of each term/word). Among the most successful weighting strategies are: the boolean weight, term frequency and relative term frequency. Next we briefly describe each one of these weighting schemes.

– *Boolean weighting:* It assigns a weight of 1 if the term t_k appears within the document d_j, otherwise the value assigned is zero:

$$w_{kj} = \begin{cases} 1, \text{ if } t_k \in d_j \\ 0, \text{ otherwise.} \end{cases} \quad (1)$$

– *Term frequency weighting (TF):* For this particular case, the assigned weight will be equal to the number of times the term t_k occurs within document d_j:

$$w_{kj} = f_{kj}. \quad (2)$$

– *Relative Term Frequency weighting (TF-IDF):* This type of weighting scheme represents a variation from the TF technique. For computing the TF-IDF

weight we must follow:

$$w_{kj} = TF(t_k) \times IDF(t_k). \tag{3}$$

where $TF(t_k) = f_{kj}$, in other words, it represents the frequency value of term t_k within document d_j. IDF is also known as the "inverse frequency" of term t_k within document d_j. The IDF value represent to some extent how "rare" is term t_k. For computing the IDF value we follow:

$$IDF(t_k) = \log \frac{|D|}{\{d_j \in D : t_k \in d_j\}}. \tag{4}$$

where D represents the document collection that is being indexed, *i.e.*, $\mathcal{D}_{Tr}$

3.3 Classifiers

Since our proposal for WSDL document representation does not dependent of a particular learning algorithm we can use any classifier to face the WS classification problem. For our experiments we selected 3 different learning algorithms which are representative of the wide diversity of methods available in the machine learning field [6, 8]. Specifically, we considered the following classifiers:

- **Naïve Bayes (NB).** A probabilistic-based method that assumes attributes are independent among them given the class.
- **Support vector machine (SVM).** A linear discriminant that aims to find an optimal separating hyperplane; a linear kernel was used for this work.
- **J48.** An algorithm used to generate a decision tree, which select the most discriminating features based on its entropy measure.

We employed the Weka implementation of the above described algorithms; default parameters were considered for all the performed experiments [7].

4 Experimental Setup

4.1 Datasets

For all the experiments performed we used a subset of the ASSAM web services collection. This collection is made up by real Web Services description documents, obtained from Salcentral [14] and Xmethods [17]. The WSDL documents are organized into a class hierarchy, that in some cases have sub-classes with at most two levels depth. Originally, this collection had 814 WSDL documents distributed in 26 classes, however, in order to prove the pertinence of our proposed approach we modified the collection applying the following steps: *i)* we flatten all classes, *i.e.,* we suit WSDL documents contained in parent classes in some particular sub-classes, and *ii)* we discarded all WSDL documents that do not contain at least one word (recognized by WordNet) for each considered *description* feature (see Table 1 for details). Due to many of WS Descriptions

have the *documentation* feature, the classification of the information extracted from this feature was taken as a baseline.

In addition to the previous collection, we also performed experiments using the OWLS-TC collection (version 3 and version 4). Together the two datasets contain more than 1000 WS, covering seven and nine categories respectively. Contrary to the ASSAM collection, the documents of the collections OWLS-TC V3 and V4 do not contain the *documentation* feature. Table 1 shows some statistics from the considered datasets.

Table 1. Basic statistics from the considered datasets.

	ASSAM	**TC-V3**	**TC-V4**
Num. docs	203	1006	1082
Vocabulary	2829	283	378
Num. classes	22	7	9
Docs by class (Avg.)	9.2	143.7	120.2
Documentation (DOC)	Yes	No	No
Terms in DOC (Avg.)	26.3	0	0

Notice that although the TC V3 and V4 are larger collections (*i.e.,* they have more WSDL documents), their vocabulary is smaller than the ASSAM collection. The reason for this difference is that, documents from the ASSAM collection do have the *documentation* feature. This characteristic can be observed also in the last row of Table 1 (*i.e., Terms in Doc*), which indicates the average number of terms for each WSDL document contained in the *documentation* feature.

It is also important to mention that the ASSAM collection represents a more complicated challenge for classifications systems, just by the fact that it contains more categories (22), and as a consequence, less examples for each class are available (9.2 in average).

4.2 Evaluation

For evaluating the classifiers performance we adopted standard measures from the text-categorization field. The leading evaluation measure is the macro F_1 measure, defined as follows:

$$Macro - F_1 = \frac{1}{|K|} \sum_{C_i \in C} \left[\frac{2 \times R(C_i) \times P(C_i)}{R(C_i) + P(C_i)} \right].$$

where the per-class recall (R) and precision (P) measures are defined as follows:

$$R(C_i) = \frac{number\ of\ correct\ predictions\ of\ C_i}{number\ of\ examples\ of\ C_i},$$

and

$$P(C_i) = \frac{number\ of\ correct\ predictions\ of\ C_i}{number\ of\ predictions\ as\ C_i}.$$

It is worth mentioning that during our experiments we applied a 10 cross-fold validation strategy.

4.3 Experimental Settings

The main goal of our experiments was to evaluate which of the information extracted from WSDL inner features improves WS classification.

Accordingly, we considered for our experiments the following *description features*: messages (*Msgs*), operations (*Names*) and types (*Params*)[6].

Hypothetically, the *Names* attribute (*i.e.,* name of a method in the WS) might be part of the selected information, which it could be more precise when includes the *Parameter* attribute (*i.e.,* names of the parameters or data types). Furthermore, if we combine these *description features* (*Names* and *Parameters*) with the *Messages* attribute we expect that it will also contain enough information to allow an automatic classifier to correctly define its category. Consequently, we defined our experiments by means of using single features and its respective combinations.

4.4 Results

The obtained experimental results are reported in Tables 2, 3 and 4 in terms of macro F_1. Results marked in **bold** indicate the *best* results obtained over different configurations.

Table 2 shows the results obtained when the ASSAM WSDL document collection is used. The first column (*i.e.*, "Description features") indicates the *description features* from which information was extracted. Notice that better results are obtained when a *boolean* representation is employed, which means that just by the presence of certain words it is possible to assign the WS category.

Table 2. Results of the experiments performed using the ASSAM collection.

Description features	BOOLEAN			TF-IDF		
	NB	**SVM**	**J48**	**NB**	**SVM**	**J48**
Msgs	0.37	0.38	0.34	0.27	0.31	0.31
Names	0.41	**0.43**	0.34	0.32	0.35	0.32
Params	0.39	0.39	0.34	0.33	0.36	**0.36**
Names+Msgs	0.41	0.41	**0.36**	0.25	0.34	0.35
Names+Param	0.42	**0.43**	0.34	**0.36**	**0.38**	0.27
Msgs+Params	0.40	0.35	0.32	0.33	0.36	0.30
Names+Msgs+Params	**0.44**	0.40	0.32	0.34	0.37	0.29

The obtained results indicate that when the *Names* feature is involved in the information extracted better results are reached. Although the best result was

[6] Refer to Section 3 to view a detailed description of the selected features.

obtained by the combination of *Names+Msgs+Params* employing a Bayesian classifier, but the combination of *Names+Params* allows a more stable behavior. This situation indicates (to some extent) that methods' names as well as the parameters have an important role on the definition of a Web service functionality, hence providing important elements to define the WS category.

Table 3 and Table 4 exhibit the results obtained on the OWLS-TC V3 and V4 collections respectively. Similarly to the results obtained on the ASSAM collection, better results are obtained under the combinations of the *Names* and *Params* description features.

Table 3. Results of the experiments performed using the OWLS-TC V3 collection.

Description	BOOLEAN			TF-IDF		
features	**NB**	**SVM**	**J48**	**NB**	**SVM**	**J48**
Msgs	0.48	0.55	0.37	0.47	0.55	0.37
Names	0.71	0.77	0.68	0.70	0.79	0.70
Params	0.73	0.78	0.79	0.62	0.80	0.80
Names+Msgs	0.71	0.77	0.68	0.69	0.78	0.70
Names+Param	0.79	**0.86**	0.82	**0.77**	0.85	0.83
Msgs+Params	0.77	0.85	**0.83**	0.73	0.85	**0.85**
Names+Msgs+Params	**0.80**	**0.86**	0.82	**0.77**	**0.86**	0.83

Notice that a better performance is obtained under a *boolean* weighting scheme. This indicates that WS categories can be determined by just the presence of certain words, which lead us to consider that counting the frequencies downgrades the classifier's performance.

Table 4. Results of the experiments performed using the OWLS-TC V4 collection.

Description	BOOLEAN			TF-IDF		
features	**NB**	**SVM**	**J48**	**NB**	**SVM**	**J48**
Msgs	0.47	0.52	0.37	0.44	0.53	0.37
Names	0.70	0.76	0.67	0.69	0.78	0.68
Params	0.74	0.79	0.78	0.65	0.80	0.81
Names+Msgs	0.70	0.76	0.67	0.69	0.75	0.68
Names+Param	**0.81**	**0.86**	**0.82**	**0.77**	**0.86**	0.84
Msgs+Params	0.78	0.85	**0.82**	0.75	**0.86**	**0.85**
Names+Msgs+Params	**0.81**	**0.86**	**0.82**	**0.77**	**0.86**	0.84

Similarly to the results obtained when the ASSAM collection is used, the combination of *Names+Params* allows to the classifier to reach an acceptable classification performance on both, the OWLS-TC V3 and V4 datasets. These particular results reinforce our intuition that extracting only information from some WSDL features can improve classification.

Finally, it is important to mention that the results obtained on the OWLS-TC V3 (Table 3) are directly comparable against results reported on [16] and [4]. On the one hand, Wang et al. [16] reports an F measure of 89% using a SVM with a lineal kernel. However, it important to remember that their proposed approach depends on the UNSPSC taxonomy (see Section 2).

On the other hand, the work proposed in [4] reports an average precision (P) of 65% and an average recall (R) of 70%. Their proposed approach, similarly to [16], depends on the UNSPSC taxonomy and WordNet (see Section 2). Although our results are reported in terms of the macro F_1, the average precision and average recall for our best configuration (*i.e., Names+Msgs+Params* using a SVM classifier in Table 3) are 87% and 86% respectively. In conclusion, our proposed approach is able to obtain a competitive performance against the state-of-the-art methods [4, 16] without needing or employing any external resource during the classification stage.

4.5 Additional Experiment

As we have mentioned in previous sections, it is believed that considering only the information contained in the *documentation* feature can improve WS classification. Accordingly, an intuitive comparison of such a system would be to contrast the obtained classification results against a system that uses the original *documentation feature* for representing and classifying WSDL documents.

On section 4.1 we showed that the only dataset that actually contains the *documentation* feature is the ASSAM collection (See Table 1). Hence, we performed an additional experiment over the ASSAM collection, where our main goal was to explore the pertinence of this particular feature when classifying WSDL documents (See Figure 2).

Notice that by using only the original *documentation feature* (*Doc*) the classification performance gets the worse results. Particularly for the Bayesian classifier, goes from a macro F_1 of 44% using the *Names+Msgs+Params* features to a 37% using only the original *documentation* feature. This result indicates, to some extent, that documentation provided by WS suppliers tends to be ambiguous and it introduces noisy elements to an automatic classification system.

Finally, it is worth mentioning that according to the *paired Student's t-test* - with a confidence level of 99 percent - the improvement obtained using *Names + Params* with both classifiers (*i.e.,* NB and SVM) are statistically significant over the results obtained employing only the *documentation feature*.

5 Conclusions

We have introduced an analysis of the combination of WSDL inner feature information that can assist to classify WS, without any extra data. These in order to select the information that improves WS categorization.

We report experiments on three different WSDL collections; the obtained results indicate that selecting only information from some WSDL inner features

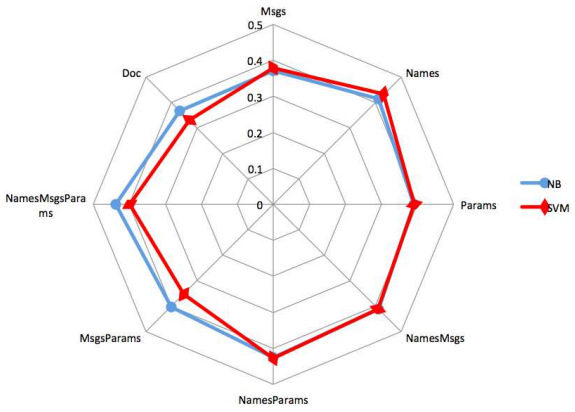

Fig. 2. Comparison of the classification performance when the *documentation* feature is employed to represent WSDL documents against using several *description* features for the construction of the proposed *virtual* feature. All the experiments were performed using the ASSAM collection.

can obtain a competitive performance against the state-of-the-art methods. The performed experiments showed that by means of combining the operations' names and parameters it is possible to obtain a macro F_1 of 86% for the OWLS-TC collections. Similar results were obtained on the ASSAM collection when we used the same combination of information extracted from WSDL features.

An additional experiment showed that the names and parameters information combination allows better classification results compared to those obtained when WSDL documents are represented by means of the *documentation* feature. A more deeper analysis demonstrate that the information from selected *description features* are able to generate a less complex and more general description of WS functionality. On the contrary, by using only the *documentation* feature results in a more complex and highly specific description of WS functionality, which leads to a less accurate classification process.

Future work directions include using Distributional Term Representations which have proven to be effective on reducing the effect of low term frequency occurrence, sparsity and term ambiguity, which are common characteristics of WSDL documents.

Acknowledgments. This work was partially funded by CONACYT under the Thematic Networks program (Language Technologies Thematic Network project 281795).

References

1. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. Addison Wesley (1999)

2. Balasubramanian, D.L., Murugaiyan, S.R., Sambasivam, G., T., V., Dhavachelvan, P.: Semantic web service clustering using concept lattice: Multi agent based approach. International Journal of Engineering and Technology 5(5), 3699–3714 (2013)
3. Batra, S., Bawa, S.: Web service categorization using normalized similarity score. Internaitonal Journal of Computer Theory and Engineering 2(1), 139–141 (2010)
4. Boujarwah, E., Yahyaoui, H., Almulla, M.: A new unsupervised web services classification based on conceptual graphs. In: ICIW 2013, The Eighth International Conference on Internet and Web Applications and Services. pp. 90–94 (2013)
5. Bruno, M., Canfora, G., Penta, M.D., Scognamiglio, R.: An approach to support web service classification and annotation. In: EEE. pp. 138–143. IEEE Computer Society
6. Duda, R., Hart, P.: Pattern classification and scene analysis. Wiley (1996), http://www.ica.luz.ve/ enava/redesn/ebooks/DHS/Versi%F3n PS/DHSChap4.ps
7. Garner, S.R.: Weka: The waikato environment for knowledge analysis. In: Proc. of the New Zealand Computer Science Research Students Conference. pp. 57–64 (1995)
8. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer (2009)
9. Liang, Q., Li, P., Hung, P., Wu, X.: Clustering web services for automatic categorization. In: Services Computing, 2009. SCC '09. IEEE International Conference on. pp. 380–387 (Sept 2009)
10. Lu, G., Wang, T., Zhang, G., Li, S.: Semantic web services discovery based on domain ontology. In: World Automation Congress (WAC), 2012. pp. 1–4 (2012)
11. Nisa, R., Qamar, U.: A text mining based approach for web service classification. Information Systems and e-Business Management 13(4), 751–768 (2015)
12. Saha, S., Murthy, C.A., Pal, S.K.: Classification of web services using tensor space model and rough ensemble classifier. In: An, A., Matwin, S., Ras, Z.W., Slezak, D. (eds.) ISMIS. Lecture Notes in Computer Science, vol. 4994, pp. 508–513. Springer (2008)
13. Sebastiani, F.: Machine learning in automated text categorization. ACM Computing Surveys 34(1), 1–47 (2002)
14. Seekda: *http://webservices.seekda.com/* (2012), last visited on September 2014
15. Stroulia, E., Wang, Y.: Structural and semantic matching for assessing web-service similarity. International Journal of Cooperative Information Systems 5(14), 407–437 (2005)
16. Wang, H., Shi, Y., Zhou, X., Zhou, Q., Shao, S., Bouguettaya, A.: Web service classification using support vector machine. In: 22nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI). vol. 1, pp. 3–6. IEEE Computer Society (2010)
17. Xmethods: *http://xmethods.net/ve2/index.po* (2013), last visited on September 2014
18. Yang, J., Zhou, X.: Semi-automatic web service classification using machine learning. International Journal of u-and e-Service, Science and Technology 8(4), 339–348 (2015)