Applying Text Classification Algorithms in Web Services Robustness Testing

Nuno Laranjeiro, Rui Oliveira, Marco Vieira
CISUC, Department of Informatics Engineering
University of Coimbra, Portugal
cnl@dei.uc.pt, racoliv@student.dei.uc.pt, mvieira@dei.uc.pt

Abstract—Testing web services for robustness is an effective way of disclosing software bugs. However, when executing robustness tests, a very large amount of service responses has to be manually classified to distinguish regular responses from responses that indicate robustness problems. Besides requiring a large amount of time and effort, this complex classification process can easily lead to errors resulting from the human intervention in such a laborious task. Text classification algorithms have been applied successfully in many contexts (e.g., spam identification, text categorization, etc) and are considered a powerful tool for the successful automation of several classification-based tasks. In this paper we present a study on the applicability of five widely used text classification algorithms in the context of web services robustness testing. In practice, we assess the effectiveness of Support Vector Machines, Naïve Bayes, Large Linear Classification, K-nearest neighbor (Lbk), and Hyperpipes in classifying web services responses. Results indicate that these algorithms can be effectively used to automate the identification of robustness issues while reducing human intervention. However, in all mechanisms there are cases of misclassified responses, which means that there is space for improvement.

Keywords—web services; robustness testing; classification

I. INTRODUCTION

Web services are now widely used to support most of the business, linking suppliers and clients in sectors such as banking and financial services, transportation, automotive manufacturing, healthcare, etc. Web services are the key element of distributed Service Oriented Architectures [1], and consist of self-describing components that can be used by other software across the web in platform-independent manner, supported by standard protocols such as SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language), and UDDI (Universal Description, Discovery, and Integration) [2]. Web services provide a simple interface between a provider and a consumer, where the first offers a set of services that are used by the second.

Robustness testing is an effective way of disclosing software bugs. It has been used mainly to assess robustness of operating systems (OS) and OS kernels [3-5], but the concept of robustness testing can be applied to any kind of interface. Robustness tests stimulate the system under testing through its interfaces submitting erroneous input conditions that may trigger internal errors or may reveal vulnerabilities.

Web services robustness testing is the adaptation of traditional robustness testing, to the web service environments. One of the first examples of robustness testing applied to web services is [6]. This paper proposes a technique to test web services using parameter mutation analysis. The web services description file (defined using WSDL) is parsed initially and mutation operators are applied to it, resulting in several mutated documents that are used to test the service. In [7] a similar approach is presented. Although it represents a more complete study, the coupling that is done to the XML (eXtensible Markup Language) technology invalidates any kind of test generalization. A more elaborated approach to assess the behavior of web services in the presence of tampered SOAP messages was proposed in [8]. It consists of a set of robustness tests based on invalid call parameters that are defined using a large set of testing rules. However, a key limitation of all web services robustness testing works is that the analysis of the testing responses is still a manual step.

Previous research has shown that web services are, in general, being deployed on the Internet with robustness problems [9]. A key problem is that, when testing web services for robustness, a very large quantity of service responses has to be manually classified in order to distinguish regular service replies from responses that indicate a robustness problem. This requires a large amount of effort and is indeed an error-prone task (due to the required human intervention). Thus, automation of this classification process is urgently needed to turn web services robustness testing into a widely used methodology.

Text classification algorithms have increasingly gained research interest in recent years and are considered to be extremely useful in the automation of several classification-based tasks (e.g., spam identification, word sense disambiguation, etc). The most common approach to text classification is based on machine learning techniques in which an inductive process automatically builds a classifier by learning the characteristics of categories from a set of pre-classified documents [10].

In this paper we evaluate the applicability of five widely used text classification algorithms in the context of web services robustness testing. In practice, we conducted a large experimental campaign in which we tested the robustness of 250 public web services, comprising 1204 operations and 4085 parameters. The 416208 web services responses obtained were manually analyzed to identify the ones that indicate robustness problems, and then used to train and exercise the text classification algorithms in a realistic scenario. Comparing the classification proposed by the different algorithms with the manually classified responses allows assess-
ing and comparing the algorithms effectiveness. An important aspect is that the algorithms used, Support Vector Machines (SVM), Naïve Bayes, Large Linear Classification (LLC), K-nearest neighbor (Ibk), and Hyperpipes, have proven in other contexts to be able to produce interesting results [10-15], thus being representative of the current state-of-the-art.

Results show that some of the algorithms are very precise in classifying robustness problems, achieving over 99% of precision using a typical training scenario (Ibk and LLC). In general, the algorithms are more precise when classifying responses that do not represent a robustness problem (i.e., are regular service responses). Hyperpipes, the third most precise algorithm, is able to outperform the top algorithms (i.e., LLC and Ibk) in classifying regular service responses that do not represent a robustness issue. Additionally, the classification process is, in general, slightly affected by the training set size variation. Large Linear Classification displays the best relation between classification effectiveness and speed of execution. As an overall conclusion, although, in general, the tested algorithms present good results, there are cases of misclassified responses. This suggests that research work is needed to further improve and tailor the algorithms to the web services robustness testing context.

The outline of this paper is as follows. Section II introduces the web services robustness testing procedure. Section III describes the experimental evaluation conducted and Section IV presents and discusses the results. Finally, Section V concludes the paper and proposes future work.

II. ROBUSTNESS TESTING PROCEDURE

The web services robustness testing approach used in this work is based on erroneous call parameters as proposed in [8]. The robustness tests consist of combinations of exceptional and acceptable input values of parameters of web services operations. These values are generated by applying a set of predefined rules according to the data type and domain of each parameter. The testing procedure consists of the following set of generic steps (see [8] for details):

1. Tests preparation: analysis of the WSDL of the web service in order to gather relevant information.
2. Workload generation and execution: execution of a workload in order to exercise the service.
3. Robustness tests generation and execution: execution of the robustness tests in order to disclose problems.
4. Web service characterization: identification of robustness problems based on the data collected before.

Before generating and executing the robustness tests we need to obtain some definitions about the web service under testing (step 1). For that, the web service interface definition, described as a WSDL file, is used to obtain the list of operations, parameters (including return values) and associated data types.

A workload (set of valid web service calls) is defined to exercise each operation of the web service under testing. The workload is generated based on the web service definitions and run without considering invalid call parameters (step 2). The goal is to understand the typical behavior expected from the web service.

Robustness tests (step 3) consist of running the workload in the presence of invalid call parameters. A set of tests is applied to each parameter (all the parameters of all the web service operations are tested). The robustness tests are automatically generated by applying a set of predefined rules (see detailed list in [8]). An important aspect is that rules focus difficult input validation aspects, such as: null and empty values, valid values with special characteristics, invalid values with special characteristics, maximum and minimum valid values in the domain, values exceeding the maximum and minimum valid values in the domain, and values that can cause data type overflow.

Figure 1. Test configuration required.

The final step (step 4) is the service behavior characterization, which includes the identification of robustness problems. The failure mode classification includes only the types of failures that can be effectively observed while conducting robustness testing of web services code from a “black-box” point-of-view. This way, based on our observations during the experimental evaluation described in the next sections, we considered the following failure modes:

- **Correct**: the web service response in the presence of an invalid input is correct (i.e., the web service responds with an expected exception or error code).
- **Crash**: the service replies with an error message or exception that indicates the occurrence of an unexpected internal problem.
- **Indeterminable**: it is not possible to determine the existence or absence of a robustness problem due to lack of information.

Figure 2 summarizes the manual procedure for the analysis of web service responses and identification of robustness problems. The terminator tags containing 0, 1, and 2 indicate respectively a Correct response, a Crash, and an Indeterminable failure. Note that, upon completion of the analysis process, a final step (not represented in Figure 2) is executed to verify if the Crash responses effectively corres-
The next step (6) is to check if the response represents an Exception that is declared in the WSDL file (declared exceptions are marked as Correct). When the exception type is not present in the WSDL and does not represent an input validation exception (7) (validation exceptions are marked as Correct), it is compared against List ψ—a list of problematic exceptions that represent typical robustness problems (8) (e.g., SQLException, NullPointerException, etc). Responses that encapsulate these types of exceptions are of course marked as Crash.

List ψ was iteratively built based on the manual analysis of more than 400 000 service responses, during our experimental evaluation, and is available at [16]. If a different exception (not present in List ψ) is observed and it clearly indicates the existence of an internal unexpected problem (9) (e.g., a username, database vendor, or filesystem structure disclosure), the response is marked as a Crash. A list of internal unexpected problems, disclosed during our large experimental evaluation, is also available at [16]. Responses that do not indicate the presence of a problem are marked as Correct, and again, uncertain cases should be marked as Indeterminable.

As mentioned before, the final step is to build a parameter view and analyze if the responses that were marked as Crash indeed represent a problematic parameter. All Crash responses are thus analyzed parameter per parameter and compared to the responses generated in the absence of injected faults. When the same reply is observed in the absence of injected faults, the response is marked as Indeterminable. In fact, some problem may exist, but it is not possible to assure that it is a robustness issue (for instance, the service may be being invoked with out of domain values - this can happen as in many cases it is not possible to know the valid domains of the parameters of public web services). In the same manner, it is not possible to distinguish which parameter is problematic if the same problem is observed when injecting the same fault in different parameters of the same operation. Indeed, even though one fault is injected at a time (in a single parameter), the fact that the domains of the parameters might be unknown, may lead to accidentally use invalid values in parameters that are not the fault injection target. These misused values may be the cause of the observed problems (instead of the injected faults), which forces the responses to be marked as Indeterminable.

III. EXPERIMENTAL EVALUATION

The goal of our experimental evaluation is to understand the applicability of existing text classification algorithms in classifying robustness problems. This section describes the tools, services, and algorithms selected and the procedure used in the experimental evaluation (including a description of the complete set of experiments executed).

A. Tools and Web Services

wsrbench is an online tool that can be used to execute robustness tests on web services and has been used to support the experimental evaluation. This tool, publicly available at http://wsrbench.dei.uc.pt, implements the web services testing approach presented previously and provides a web-based interface that allows users to configure, execute tests, and also visualize the tests results. wsrbench is free, open-
source, and very easy to use, requiring only a simple registration and posterior authentication process.

Using wsrbench, we tested the robustness of 250 public web services, comprising 1204 operations and 4085 parameters, deployed over 44 different country domains, and provided by 150 different relevant parties. These parties include several well known companies like Microsoft, Volvo, Nissan, and Amazon; multiple governmental services; banking services; payment gateways; software development companies; internet providers; cable television and telephone providers, among many others. The complete list of tested services includes web services deployed on 17 distinct server platforms and 7 different web service stacks. A complete and detailed list can be found at [16].

The selected services were obtained using Seekda (http://webservices.seekda.com/) – a web service search engine. This is, to our best knowledge, the largest web service search engine currently available on the Internet. The service selection process consisted in introducing technology-related keywords (e.g., web, service, xml, etc) in the search engine and randomly selecting some services from the search results.

It is important to note that due to the large heterogeneity of the tested services, we obtained service responses in several languages (e.g., Chinese, Russian, etc). In order to respect the random selection process and have a richer service sample, we translated these responses from their original language to English in a pre-processing step (we built a tool that executes this translation in an automatic way by using Google translate [16]). This allowed us to not discard services that would otherwise become impossible to understand and analyze. Note that in order to evaluate the algorithms effectiveness we have to analyze each response manually. In this sense, this pre-processing step is crucial.

As a support tool for the execution of the classification algorithms, we chose Weka [17]. It is a well-known tool that provides a comprehensive collection of machine learning algorithms and data pre-processing tools in an integrated and easy to use environment.

B. Classification Algorithms

In this section we summarize the classification algorithms used in the study. Interested readers may find more detailed descriptions in the references provided.

The Naive Bayes algorithm is a probabilistic classifier based on the Bayes' theorem [18]. This algorithm is known by its speed and good performance on text classification. However it assumes that all feature words are independent which has a penalty on the end results.

The SVM classifier has been used in a wide variation of text classification problems [19]. It builds a hyperplane (or set of hyperplanes) in a high or infinite dimensional space that predicts whether a new example falls into one of the available categories [20].

Hyperpipes is a simple and fast classifier, adequate to scenarios using a large number of attributes [11]. For each class of the dataset, a structure (hiperpype) is constructed with all the points of that class. Instances are classified based on the class that has the most similar instance [21].

The K-nearest Neighbor is an instance-based algorithm. Decision rules are used to assign to an unclassified sample point the classification of the nearest of the previously classified points [22]. This algorithm as shown to be effective in a variety of problem domains, including text classification [12,13].

Large Linear Classification is an algorithm that supports logistic regression and linear support vector machines, and is adequate to large sparse data with a huge number of instances and features [23]. This algorithm is a variant of the traditional Linear Classification algorithm, and has proven to have superior classification rates, requiring short times for learning and classification [14]. Its behavior is similar to SVM, and consists of finding hyperplanes that approximately separate a class of document vectors from its complement [24].

Several previous works have been conducted involving these five classifiers. A work on the field of visual and linguistic information [11] has proven that Hyperpipes can have an accuracy and performance greater than Naive Bayes and SVM. In the field of feature selection for text classification, the work from Rogati et al [15] shows that variants of KNN and SVM remain very competitive, showing both better performances than Naive Bayes. In [14] it is shown that the Large Linear classification can outperform SVM in terms of performance with a smaller training time.

C. Experiments Description

In order to evaluate the effectiveness of the classification algorithms we executed the following sets of experiments:

Set i. Execution of the robustness tests and manual classification of robustness problems;
Set ii. Experimental evaluation of the effectiveness of the five classification algorithms;
Set iii. Analysis of the impact of the training set size in the two most effective algorithms;
Set iv. Comparison of the five algorithms in terms of the relation between their effectiveness and speed.

The first set of experiments (Set i) consisted in executing the robustness tests using wsrbench and analyzing each test result (i.e., each service response) individually. This manual analysis was done according to the procedure described in Figure 2 and we double-checked each test result (a total of 416208 responses) in an effort to reduce the possibility of human error (this process was done by two distinct and independent software developers). The goal was to build baseline information to be used in the evaluation of the classification algorithms in the following experiments.

The second experimental set (Set ii) consisted of testing the five algorithms. Generally, text classification tasks involve two steps: building a model from a training set and using that model as a basis for the classification process itself. Thus, the first step was to select the size of the training set and define an approach for selecting the data for training (leaving the remaining for classification). As the goal is to understand the general classification capabilities of each of the different algorithms, we opted to use 30% of the total
dataset as training data. It is important to indicate that this is a value frequently used in the field and is accepted as adequate for generic classification tasks [25]. Each record of the training set (extracted from the wsrbench output) contained the service URL, the fault type (applied during the corresponding robustness test), the SOAP request, the service SOAP response, and a field with the result of the manual classification procedure. To avoid introducing an artificial bias that could favor the classification process, we used a random strategy to select the training data from the complete experimental dataset.

Using these training definitions as reference we executed each of the five classification algorithms ten times (a total of 50 executions). Each execution used a new random training set and we opted to use the algorithms default values during the various experimental sets. Among the default values used, we highlight the following: the use of a total number of 1000 word features for the classification process; C-Support Vector Classification as SVM type (and radial basis function as kernel function); no smoothing method for Naïve Bayes; k=1 for the Ibk algorithm; and L2-loss support vector machines as solver for the LLC algorithm. We opted to use the default values as we had indication of good results in other classification contexts [11,26]. This set of experiments enabled us to collect rich data and moreover, it enabled us to select the two best algorithms and use them to understand the relation of their effectiveness with the training set size and also with their execution speed.

In the third set of experiments (Set iii) the two most effective algorithms (Ibk and Large Linear Classification) were used to understand the impact of the training set size. We opted not to test the remaining classification algorithms as they had already shown poorer results and, most likely, decreasing the training set would not improve the scenario. We used three additional training set sizes of 1%, 10%, and 20%. We executed ten runs for each training size, using randomly selected data (we had already tested the 30% size in experimental Set iii). This way, for Ibk and Large Linear Classification we have a total of 80 executions using 1%, 10%, 20%, and 30% as sizes for the random training set. Although larger sizes could yield better results, generally smaller training sets are a more adequate option, as it is usually more difficult and time consuming to collect large datasets for training purposes.

A final set of experiments (Set iv) was conducted to understand the relation between the algorithms effectiveness and their speed of execution. For this purpose, we used the same experimental configuration as in Set ii, that is, 50 executions (10 for each algorithm) using 30% as size for the training set. We then extracted the training and classification times and used them to understand the relation between the execution time and the algorithms effectiveness.

The tool used for executing the tasks involved in the experimental evaluation in an automatic way (selecting training data, executing an algorithm, etc) is publicly available at [16].

IV. RESULTS AND DISCUSSION

This section discusses the results of the experimental evaluation. Results are analyzed individually for each set of experiments (as presented in section III.C).

A. Overall Results from Manual Analysis

This section presents the results obtained for experimental Set i. Figure 2 presents the global results for the public services evaluation in three different granularity perspectives. The service execution granularity presents the analysis from the service perspective, being the service the unit of analysis. Similarly the operation and parameter granularity consider these two items the analysis unit. As we can see, results indicate that a large number of services are currently being deployed and made available to the general public with robustness problems. In fact, 46% of the tested services presented some kind of robustness issue. This is a very large percentage of problematic services and the problem gains a larger dimension if we consider that a large part of these problems may also represent security issues.

If we consider the ‘parameter’ execution granularity, we can see that the ‘Correct’ failure mode was observed at least once for 77% of the parameters. In fact, the correct behavior is present in most of the parameters tested, which indicates that at some point services are able to display an adequate and expectable behavior. However, there are still relatively high percentages of the ‘Crash’ and ‘Indeterminable’ failure modes, respectively 15% and 7%. These are obviously non-additive results as the same service can display multiple failure modes.

We can also observe that the percentages of the different failure modes generally decrease as we increase the analysis granularity. Note that each failure mode only needs to be observed once (in a given parameter), so that we mark the whole parameter, operation, and service with that failure mode. This justifies the fact that higher execution granularities generally display higher percentages for each failure mode. Despite this, the global image is maintained being the ‘Correct’ and ‘Indeterminable’ failure modes, respectively the most and least observed modes, whereas the ‘Crash’ failure mode consistently maintains its middle position.

![Service behavior by execution granularity](image)

Figure 3. Global robustness results.
B. Analysis of the Algorithms Precision

This section presents the results obtained for experimental Set ii. Figure 4 shows the overall results for the precision of the five algorithms tested. The tabular version presents detailed results that include both absolute and percentage values (including the standard deviation observed – columns Stdev). Considering these global results, it is clear that Ibk and Large Linear Classification algorithms are the ones that exhibit the best results, being able to classify correctly approximately 99% of the web services responses. Hyperpipes and Support Vector Machines closely follow the best two algorithms, with an overall precision of approximately 97%. On the other hand, Naïve Bayes delivered the worst results by being able to classify correctly only 90% of the total results.

Figure 4. Overall results for the algorithms precision.

Precision of the classification results

1) Precision of the classification results

Figure 5 presents a detailed view of the precision of the five algorithms considering each one of the three failure modes. In general, the algorithms display better results for the Correct failure mode and poorer results for Crash and Indeterminable. This can be explained by the fact that these two failure modes are not as frequent as the former one. Thus, with a random sample, fewer cases will be observed during the training phase and this may explain the lower classification precision. Still considering the Correct failure mode, four out of five algorithms present very good results (more than 99%) and Hyperpipes manages to classify 99.99% of these records. The exception to this general picture is Naïve Bayes with approximately 94%.

There is a lot more diversity for the Crash failure mode results. In this group of results, Support Vector Machines is the worst performer classifying correctly only approximately 64% and Larger Linear Classification and Ibk present the best results displaying 96% and 93%, respectively. Concerning the Indeterminable failure mode, Ibk and Large Linear Classification present again the top results (96% and 92%, respectively). However, Support Vector Machines is now the third best algorithm with 92% of precision. Hyperpipes and Naïve Bayes have the worst results as in the previous group.

2) Analysis of the misclassified responses

Figure 6 presents a global view of the distribution of the misclassified responses (i.e., responses wrongly classified by the algorithms) per failure mode for each algorithm. The percentages were calculated using the total misclassified results as basis. When observing the distribution of incorrect results in detail, we can see that Hyperpipes presents poor results for the Indeterminable failure mode (65% of the total responses misclassified by this algorithm) and Support Vector Machines is a better classifier with only 26% of incorrect classifications. However, we can see that for Ibk 42% of the total incorrectly classified responses belong to the Indeter-
3) Analysis of how the algorithms fail

Figure 7 presents the misclassified results in detail. Graphical and tabular views are provided as before. Each table column header represents a pair of failure modes (the manual classification and the result of the automatic classification), or in other words, a misclassification type. For instance, the first column represents the total average of responses that were manually classified as Correct and were classified as Crash by the algorithms. Thus, we can observe how each algorithm failed in each of the three failure modes. Values in bold indicate the best value observed (i.e., lowest count) for each type of misclassification.

As already mentioned, Hyperpipes is the algorithm that fails the less in the Correct failure mode. In fact, it presents the lowest averages for both possible cases (a Correct answer classified as Crash or classified as Indeterminable). Correct responses classified as Crash or Indeterminable do not represent a major issue. In fact, in a real scenario, these two failure modes have to be manually analyzed by the developer (in order to fix potential robustness issues). These false positives would represent some additional work, but the effort should be low as it includes a very small number of cases.

When considering responses manually classified as Crash, the scenario is slightly different. Hyperpipes is still the best algorithm when a Crash response was classified as Indeterminable (in table column ‘Crash->Indeterminable’) by the algorithm. However, when the error was the classification of a Crash response as Correct response, the Large Linear Classification algorithm displays the lowest average
(see table column ‘Crash—Correct’). Although Crash responses classified as Indeterminable are not a problem (again, the developer has to analyze them in order to fix the potential problems), Crash responses that are classified as Correct are a major issue. In fact, a key goal of using these algorithms is to avoid requiring the developer to manually analyze the responses classified as Correct (in order to reduce effort). This way, the responses misclassified as Correct will, in principle, not be subject of further analysis, which may lead to robustness problems not being fixed.

Finally, for the responses manually classified as Indeterminable, Ibk is now the algorithm that presents the best absolute results. In all algorithms there are some Indeterminable responses that are classified as Crash and Correct. While the first is not a major issue, the second might lead again to potential robustness problems not being fixed.

C. Analysis of the Impact of the Training Set

This section presents the results obtained for experimental Set iii. Figure 8 shows the impact of the training set size on the precision of the classification in graphical and tabular views. Generally, the impact is minimal, suggesting that the algorithms tested do not need large training sets in order to achieve better results. In fact, the tabular view indicates that the global precision varies from 99.25% to 98.06% in Large Linear Classification, and from 98.95% to 98.36% in Ibk. These minimal variations are particularly visible in Correct responses. In all cases the differences between the 4 training set sizes is negligible (less than 0.2% between the worst and best observed result).

The Crash failure mode displays a larger variation. The Large Linear Classification algorithm loses approximately 1% of its precision when the training set size is decreased from 30% to 10%. However, decreasing the training set size from 10% to 1% results in a 9% drop of its precision. The Ibk algorithm produces consistent results in the 10%-30% sets (from 93.2 to 93.6). Concerning the 1% set, it loses 7% of its precision when compared with the 10% set.

The results for the Indeterminable failure mode are very stable. They vary 0.5% for the Large Linear Classification algorithm and 0.3% for Ibk in the 10%-30% range. The total observed variation, for this failure mode and considering all training sets, is approximately 3.3% and 1.1%.

As referred, the process seems to be relatively unaffected with the decrease of the training set size. A slight exception may be the Crash responses for the 1% for the training set. Indeed, Crash failures appear to add more difficulties to the algorithms when the training set is smaller. Potential reasons for these classification issues may be the diversity of responses that indicate robustness problems (resulting from the heterogeneity of the services) and also their smaller relative frequency (comparing to correct web service responses). Despite this, the classification precision remains, in general, high for all failure modes.

Figure 9 presents the impact of the training set size variation on the responses that are incorrectly classified. The graphic view displays percentages calculated using the total incorrectly classified responses as basis. The tabular view presents the absolute values observed.

Concerning the Ibk algorithm, there is not much variation in the relative percentages of each failure mode. However, we can see that decreasing the training size to 1% results in a larger relative percentage of Crash responses being incorrectly classified and the same happens when we consider the absolute values. On the other hand, both Correct and Crash responses in the 1% experiments decrease their relative frequency. Despite this, we can see that, in all cases, the absolute values of misclassified responses increase with the reduction of the training set.

Considering the Large Linear Classification algorithm, as

![Figure 8. Training set size impact in the algorithms precision](image)

![Figure 9. Impact of the training set size in the misclassification](image)

![Figure 10. Absolute Values](image)
we decrease the training set size, Crash responses incorrectly classified increase in both percentage and absolute terms. On the other hand, responses in the Correct class decrease in percentage values, but however increase in absolute values. More variation exists in the absolute values for the Indeterminable responses, which however experience a larger increase when the training set size is reduced from 10% to 1%.

D. Relation between Effectiveness and Speed

In this section we present the results obtained with experimental Set iv. As referred, the goal is to understand the relation between the algorithms precision and their execution speed. Table I presents the execution times for each of the five algorithms. The table presents the average execution times (and standard deviation) for both training and classification phases (using 30% and 70% of the training data, respectively). It also presents the relation between the training time and classification time (in percentage).

Considering the absolute averaged values presented in Table I, it is easy to observe that Ibk and Support Vector Machines are much slower in the complete process, consuming over three and four hours, respectively. The remaining algorithms required less than one hour to execute. Hyperpipes in particular was very fast, taking under 15 minutes to conclude the whole operation.

An interesting aspect is to understand the relation between the training time and classification time. In fact, Ibk was able to train very fast (the process took, in average, less than 4 minutes, corresponding to 1% of the classification time). On the other hand, we found the higher relation between the training and the classification process for the Large Linear Classification algorithm. Training was in fact slower in this case, taking approximately 15 minutes, which corresponds to 153% of the total classification time.

Table II presents a sorted view of the algorithms considering their execution times and precision. As already shown, Hyperpipes offered the best performance in terms of execution speed. However, in terms of precision it is placed in third place (closely followed by the fourth place holder), which may not be useful, depending on the tester’s preferences (developers/testers frequently prefer precise results).

## Table I. Execution times for the classification algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Large Linear Class.</th>
<th>Naïve Bayes</th>
<th>SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Stddev</td>
<td>Time</td>
</tr>
<tr>
<td>Train</td>
<td>0:02:24</td>
<td>0:00:18</td>
<td>0:14:16</td>
</tr>
<tr>
<td>Classification</td>
<td>0:12:10</td>
<td>0:02:18</td>
<td>0:09:18</td>
</tr>
<tr>
<td>Train/Class.</td>
<td>20%</td>
<td>-</td>
<td>153%</td>
</tr>
<tr>
<td>Total</td>
<td>0:14:34</td>
<td>0:02:22</td>
<td>0:23:35</td>
</tr>
</tbody>
</table>

The second fastest is the Large Linear Classification algorithm. In fact it took, in average, less than 24 minutes in the complete process and still it was the most precise algorithm, classifying more than 99% of the cases in a correct way. If we observe the results from the precision point of view, Ibk also presents results close to the best observed (~99%). However it was the slowest of all five algorithms taking longer than four hours in the complete process.

V. Conclusion and Future Work

This paper presented an experimental study on the applicability of five widely used text classification algorithms (Support Vector Machines, Naïve Bayes, Large Linear Classification, Ibk, and Hyperpipes) in the context of web services robustness testing. A large experimental evaluation has been performed and results indicate that:

- Ibk and Large Linear Classification algorithms are very precise in classifying robustness problems, achieving over 99% of precision using a typical training scenario. The worst precision performance is obtained with Naïve Bayes. This algorithm was able to correctly classify approximately 90% of the service responses;
- In general, the algorithms are more precise when classifying responses that do not represent a robustness problem (i.e., regular service responses). In fact, four out of the five algorithms displayed a precision for Correct responses equal or higher than 99.75%. This means that these algorithms can be extremely useful in reducing the set of responses that the developer has to analyze, to fix existing robustness issues;
- Hyperpipes, the third most precise algorithm, is able to outperform the top algorithms (i.e., Large Linear Classification and Ibk) in classifying regular service responses that do not represent a robustness issue;
- The impact of the training set size variation in the classification process is, in general, very low, which suggests that the performance of the algorithms in presence of new cases is good. Additionally, these algorithms can be extremely useful even in the cases where the training basis is small (a frequent case);
- The algorithms execution times are highly variable, ranging from approximately 15 minutes in Hyperpipes to more than 4 hours in Ibk. Nevertheless, these times seem quite acceptable taking into account the large set of responses classified (a total of 416208 records for training and classification), which suggests that time is not a issue regarding the

## Table II. Execution speed and precision comparison

<table>
<thead>
<tr>
<th>Order</th>
<th>Execution Speed</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Algorithm</td>
<td>Value</td>
</tr>
<tr>
<td>1º</td>
<td>Hyperpipes</td>
<td>0:14:34</td>
</tr>
<tr>
<td>2º</td>
<td>Large Linear Class.</td>
<td>0:23:35</td>
</tr>
<tr>
<td>3º</td>
<td>Naïve Bayes</td>
<td>0:57:59</td>
</tr>
<tr>
<td>4º</td>
<td>SVM</td>
<td>3:24:37</td>
</tr>
<tr>
<td>5º</td>
<td>Ibk</td>
<td>4:26:17</td>
</tr>
</tbody>
</table>
adoption of this type of algorithms in the web services robustness testing scenario.

- Large Linear Classification clearly displays the best relation between classification effectiveness and speed of execution;
- As a final note, it is possible to refer that, in general, the results of our experimental evaluation also match results observed in the previous works presented earlier at the end of Section III.B.

Although some algorithms can produce interesting results in the classification of robustness issues, there is still margin for improvement. For example, avoiding the misclassification of responses that indicate robustness issues as correct responses is of utmost importance. In fact, although overall results suggest that we can rely in some of the algorithms tested, none is 100% precise, which affects the robustness problems detection coverage and also opens space for the misidentification of inexistent problems (i.e., false positives).

After analyzing the results, we can see that it may be possible to improve the overall performance by trying to use the best of each algorithm. In fact, as future work, we intend to study ways of effectively combining the output of each algorithm and perform a more elaborated pre-processing of the algorithms input data (i.e., web services responses). Additionally, we intend to create a custom implementation of a new classifier algorithm (based on the manual classification process) and compare its performance with existent algorithms. These efforts can be very helpful in providing better overall results, and we may be able to provide developers with a precise and easy way of classifying robustness issues.

REFERENCES