Harnessing the Crowds for Automating the Identification of Web APIs

Carlos Pedrinaci and Dong Liu and Chenguai Lin and John Domingue
Knowledge Media Institute
The Open University
Walton Hall, Milton Keynes, MK7 6AA, UK

Abstract
Supporting the efficient discovery and use of Web APIs is increasingly important as their use and popularity grows. Yet, a simple task like finding potentially interesting APIs and their related documentation turns out to be hard and time consuming even when using the best resources currently available on the Web. In this paper we describe our research towards an automated Web API documentation crawler and search engine. This paper presents two main contributions. First, we have devised and exploited crowdsourcing techniques to generate a curated dataset of Web APIs documentation. Second, thanks to this dataset, we have devised an engine able to automatically detect documentation pages. Our preliminary experiments have shown that we obtain an accuracy of 80% and a precision increase of 15 points over a keyword-based heuristic we have used as baseline.

1 Introduction
Service-orientation prescribes the development of software applications by reusing (possibly remote) services, that is, software components offered via programming-language independent interfaces. On the Web, service technologies are currently marked by the proliferation of Web APIs, also called RESTful services when they conform to REST principles (Fielding 2000). Major Web sites such as Facebook, Flickr, Salesforce or Amazon provide access to their data and functionality through Web APIs. This trend is largely driven by the simplicity of the technology stack as well as by the rapidity with which third parties are combining diverse APIs into mashups that provide added-value solutions.

Unfortunately, Web APIs are most often described solely through HTML Web pages that are intended for humans and provide no convenient means for supporting their automated discovery, invocation, and composition (Maleshkova, Pedrinaci, and Domingue 2010). For instance, in order to use Web APIs we now need to rely entirely on humans with regards to the identification and interpretation of the API descriptions since general-purpose Web search engines cannot distinguish Web pages documenting APIs from, e.g., tutorials, blogs and Questions & Answers sites about software.

Based on the manual contribution of hundreds of users, ProgrammableWeb has established itself as the main public registry for Web APIs. Although highly valuable, ProgrammableWeb provides limited discovery capabilities and sometimes presents information that is out of date or provides incorrect links to APIs documentation pages. As a consequence, it is still necessary for developers to go through the results obtained, filter those that are not valid or relevant, locate and eventually interpret the Web APIs documentation. Furthermore, it may even be necessary to browse the Web in case a better option exists and it has not yet been added to the registry.

In this paper we present our preliminary results towards the development of an automated Web API search engine. In particular, we present our current work on a targeted Web crawler for Web APIs which combines crowdsourcing techniques with machine learning algorithms. This paper provides two main contributions. First, we have produced a curated dataset of Web APIs documentation and the related infrastructure to populate it. The infrastructure exploits both indirect crowdsourcing techniques by obtaining users’ generated data from ProgrammableWeb, and direct crowdsourcing by means of a Web application that has been developed for enabling users to easily assess if a given Web page documents an API or not. Second, on the basis of this dataset, we have trained an engine which is able to detect Web APIs with an 80% accuracy and an overall precision of 75%.

The remainder of this paper is organised as follows. In Section 2, we present background information and related work on Web APIs and service discovery in general. In Section 3 we present the overall approach we are pursuing and introduce the main details behind the machine learning algorithms adopted for the Web API identification engine. We then present both the approach we have followed for training the engine and cover the details of our curated dataset of Web API documents. Finally, we describe some preliminary evaluation results, present the main conclusions drawn from this work and introduce the future research we are planning to carry out.

2 Background and Related Work
A fundamental tenet of service-oriented software is the discovery of services, may it be at runtime or at design time. Service discovery enables the development of software by
reusing existing components, the integration of functionality provided by third-parties, or even the adaptation of software to changing conditions such as the user context, execution errors, etc. Thus far service registries have been the main approach adopted in order to provide programmatic access and discovery of services, establishing themselves as the meeting point between clients and service providers.

Up to this date, work on service discovery has focused on the most part on WSDL Web services, the most renown example being Universal Description Discovery and Integration (UDDI) (Erl 2007; Pedrinaci, Domingue, and Sheth 2010). UDDI’s success was limited, however, as these registries were relatively complex and yet did not support expressive enough queries (Pilioura and Tsalgatidou 2009). Today, Seekda\(^1\) provides one of the largest indexes of publicly available Web services, currently containing 28,500 Web services with their corresponding documentation.

Semantic Web services (SWS) researchers have long tried to overcome the limitations of service technologies by using semantic descriptions. A number of models, discovery engines, and further supporting infrastructure have been developed over the years (Pedrinaci, Domingue, and Sheth 2010). Much research has been devoted to improving the performance or expressivity of discovery algorithms, and also to exploiting machine learning and information retrieval techniques to bring additional flexibility to the matchmaking process (Kawamura et al. 2004; Toch et al. 2007; Stollberg, Hepp, and Hoffmann 2007; Kiefer and Bernstein 2008; Klusch, Kapahnke, and Zinnikus 2009). Nevertheless, despite demonstrating the advantages of semantic annotations in discovering services, the vast majority of the SWS initiatives are predicated upon the semantic enrichment of WSDL Web services, and these have turned out not to be prevalent on the Web where Web APIs are increasingly favoured (Pedrinaci and Domingue 2010).

In order to better understand what the main characteristics of existing Web APIs are and how we can better support their use on a Web scale, we previously carried out a thorough analysis over a body of publicly available Web APIs (Maleshkova, Pedrinaci, and Domingue 2010). In the survey we took a random selection of more than 200 APIs listed at ProgrammableWeb. For each of these APIs we tracked general information (e.g., number of operations, number of mashups), the type of Web API (RESTful, RPC-oriented, or hybrid), information about the inputs and outputs (representation formats, parameter types), invocation details and some further details such as the presence of example requests. This survey highlighted among other results, see (Maleshkova, Pedrinaci, and Domingue 2010), that:

- Most Web APIs are solely described in highly heterogeneous HTML documents.
- The quality of the descriptions leaves a lot to be desired. For instance, only 28% of the APIs clearly state the data-type of their parameters, and only 60% state the HTTP method to be used.
- both REST and RPC-oriented APIs are provided but the vast majority still tend to provide RPC-oriented APIs (only about 32% of the Web APIs appeared to be RESTful).

In the light of the lack of standardisation in this area, there have been proposals for using languages like WADL\(^2\) and WSDL 2.0\(^3\) for providing Web API descriptions which could help support discovering and using Web APIs. Yet, their adoption has been minimal and providers still prefer using custom-tailored HTML documents instead. Driven by this fact, some researchers have proposed the annotation of these Web pages as a means to support Web APIs discovery and invocation. The main proposals in this respect are SA-REST (Sheth, Gomadam, and Lathem 2007) and hRETS/MicroWSMO (Kopecky et al. 2011) which provide means for identifying services, operations, endpoints as well as they enable attaching semantic annotations. Although their benefit has been highlighted in a number of occasions these academic initiatives are still to gain further adoption.

From the perspective of supporting the discovery of this newer kind of services, there has not been much progress. Perhaps the most popular directory of Web APIs is ProgrammableWeb which, as of January 2012, lists about 4,800 APIs and 6,400 mashups. This directory is based on the manual submission of APIs by users and currently provides rather simple search mechanisms based on keywords, tags, or a simple prefixed classification, none of which are particularly expressive. Based on the data provided by ProgrammableWeb, APIHut (Gomadam et al. 2008) increases the accuracy of keyword-based search of APIs compared to ProgrammableWeb or plain Google search. Finally, iServe (Pedrinaci and Domingue 2010) enables the application of advanced (semantic) discovery algorithms for Web API discovery but, thus far, it is limited by the fact that it relies on the presence of hRETS annotations in Web pages which are still seldom available.

A fundamental drawback of ProgrammableWeb and by extension of APIHut is that they rely on the manual registration of APIs by users. As we shall illustrate in this paper, the data held tends to be out of date (e.g., APIs that have been discontinued are still listed) and often provide pointers to highly generic Web pages (e.g., the home page of the company offering the API) which are not particularly useful for supporting the discovery and use of the related APIs. Despite the increasing relevance of Web APIs there is therefore hardly any system available nowadays that is able to adequately support their discovery. The first and main limitation in this regard concerns the automated location of Web APIs, which is the main focus of this paper.

3 Automating the Identification of Web APIs

Given that the main means used for publishing and describing Web APIs are plain Web pages, in our ongoing work on iServe, a public platform for service discovery, we are approaching the discovery of Web APIs as a targeted Web

---

\(^1\)See http://webservices.seekda.com/

\(^2\)http://www.w3.org/Submission/wadl/

\(^3\)http://www.w3.org/TR/wsd20-primer
The NB classifier is a probabilistic model that assumes independence among its features. Given a document $d_t$, the task is to calculate the posterior probability that it belongs to class $c_j$ using

$$P(c_j|d_t) = \frac{P(c_j)P(d_t|c_j)}{P(d_t)}$$

where $P(c_j)$ is the prior probability of class $c_j$, and $P(d_t|c_j)$ is the likelihood of document $d_t$ given class $c_j$.

**Naive Bayes** The NB classifier is a probabilistic model which embodies strong independence assumptions about how the data is generated. Specifically, NB assumes that given a class label, all attributes of the examples are independent of each other, which greatly simplifies the calculation of joint distribution over the attribute space (Lewis 1998). In a standard text classification task, some labelled examples are provided as training data, based on which the optimal parameters estimates of the NB classifier are calculated. The trained model based on these estimates can then be applied to the test set to determine which class the unseen example most likely belongs to.

Assume that we have a corpus with a collection of $D$ documents denoted by $D = \{d_1, d_2, ..., d_D\}$, and each document is accompanied by one of the binary class labels $C = \{c_0, c_1\}$ indicating whether the document describes a Web API or not. Given a vocabulary with $V$ distinct terms denoted by $\{1, 2, ..., V\}$, each document can then be presented as a $V$-dimensional binary vector, with each of the dimensions $t$ corresponding to word $w_t$. With the binary independence setting, $\lambda_{it}$, the $t$th dimension of document $d_t$ can only take two possible values, i.e., 0 or 1, indicating whether word $w_t$ has occurred at least once in the document.

When predicting the class label of a test document $d_t$, the goal is to calculate the posterior probability that the unseen document $d_t$ is generated from class $c_j$ using

$$P(c_j|d_t) = \frac{P(c_j)P(d_t|c_j)}{P(d_t)}$$

where $P(d_t|c_j)$ is the likelihood of document $d_t$ given class $c_j$.

**Support Vector Machines** SVMs have been shown to be highly effective in various text classification problems (Cortes and Vapnik 1995). The basic idea behind SVM is to find a classification decision boundary, for which the margin, defined as the smallest distance from the decision boundary to any of the training examples, is as large as possible. In contrast to probabilistic classifiers such as naive Bayes, SVM is a discriminative model which does not provide posterior probabilities. Instead, the classification for unseen data in SVM depends only on the kernel function evaluated on a subset of training data points which are the so-called support vectors.

Given a training set comprising $D$ document-label pairs $(x_i, c_i)$, where $x_i$ is a vector and $c_i \in \{-1, 1\}$ corresponding to true and false class respectively, the SVMs require the solution of the following optimisation problem:

$$\min \frac{1}{2} W^T W + L \sum_{i=1}^{D} \epsilon_i$$

subject to $c_i (W \phi(x_i) + b) \geq 1 - \epsilon_i$, where $\phi(x_i)$ is a fixed feature-space transformation, $b$ is the bias parameter, $\epsilon_i$ is the slack variable and $L > 0$ is a trade-off parameter.
Web tool called API Validator\(^5\) which loads Web pages through an iframe and provides a simple interface allowing one to quickly assess if a Web page is describing a Web API or not. The tool is thought for exploiting the feedback of users distributed all over the world in order to easily generate a curated dataset of URLs of Web pages documentation.

### 4.1 Dataset Generation

The current version of API Validator is informed by 3,600 URLs provided by users to ProgrammableWeb obtained in May 2011. On the basis of these URLs, the tool gradually explores the dataset presenting the user with one Web page at a time. For each Web page the user can indicate if the Web page describes an API or not. The tool additionally presents the possibility for users to skip a Web page should the answer not be clear.

Each time a Web page is assessed API Validator enriches its dataset with this information. The tool supports establishing a threshold of the level of agreement across several evaluations of the same Web page in order to consider the resulting assessment as definitive. For the purposes of this paper, the tool was configured to only require one assessment to speed up the evaluation. A more thorough evaluation involving several assessments per API is planned in order to ensure that we do not have errors in the data as well as to better deal with controversial cases.

In order to generate a dataset for our experiments, the first three authors of the paper used the tool to process 1,871 Web pages, i.e., over 50% of the entire dataset. Pages were accepted as being Web APIs documentation whenever the main purpose of the page was to document an API. This also includes for instance index pages like the Last.fm page in Figure 1. Whenever the Web page was down or the assessment was not clear we skipped it.

Statistics about the resulting validation dataset\(^6\) are presented in Table 1. In total we validated 1,553 Web pages listed in the ProgrammableWeb dataset, which constitutes the 43.14% of the 3,600 URLs we started with. Out of these URLs, we manually classified 624 Web pages as documenting a Web API, and 929 as not documenting a Web API. Additionally, 318 pages were skipped because either the server was down or the assessment was not clear. One noticeable characteristic of the validated Web APIs dataset is that examples from different classes are imbalanced (624 Web pages documenting APIs vs. 929 normal Web pages). The reader should note, however, that on the Web this imbalance also exists and is even more accentuated.

Our manual validation shows clearly the issues we introduced earlier on concerning the limitations of the data gathered by ProgrammableWeb. The reader should note, however, that these figures indicate that the URL listed by ProgrammableWeb as corresponding to the Web API in many cases is not a documentation Web page. This does not mean that there does not exist an API at that Web site. We have indeed seen that this is true in certain cases but developers would still need to dig for the API documentation which again highlights the need for better automated solutions.

After the validation of the Web pages, the dataset was pro-

\(^5\)See http://iserve-dev.kmi.open.ac.uk/validator/

\(^6\)The validation dataset can be found at http://iserve.kmi.open.ac.uk/
cessed in order to extract the main features of each HTML document. In the preprocessing, we first cleaned up the Web pages using the HTML Tidy Library\(^7\) to get rid of issues in the HTML. An HTML parser was subsequently used to extract contents from HTML pages. At this stage we currently discard the HTML tags although it is expected that keeping track of this information could help us reach better results. Additionally, any scripts contained, i.e., content within the \\[	ext{<script}>\text{ tag, was also removed. In the second step, wildcards and word tokens with non-alphanumeric characters were removed and all word tokens in the dataset were lower-cased. After preprocessing, we obtained a total number of 54,299 words in the vocabulary of the corpus.

5 Evaluation Results

Using this dataset we trained the Web API identification engine. We used both the NB classifier from Weka\(^8\) and an SVM implementation based on the libSVM\(^9\) package. For both classifiers, all parameters were set to the default values except that linear kernel function was used for SVM.

In order to better assess the performance of the engine given the fact that there currently does not exist a comparable engine other than, to some extent, ProgrammableWeb which was used for the generation of the training data, we implemented as baseline a keyword-based heuristic. The heuristic is based on the co-occurrence of keywords within a Web page. In particular, this heuristic classifies as Web API documentation any Web page where the number of different keywords that occur is equal or greater than a threshold. For our experiments the list of words used was api, method, operation, input, output, parameter; get, post, put, delete, append, url—which are known to occur often—and the optimal value for the threshold (i.e., 3) was chosen. Confusion matrices summarising the number of instances predicted correctly or incorrectly by each of the methods are shown in Table 2.

Table 3 shows the classification results of the tested classifiers with different evaluation measures based on a 5-fold cross validation. The keyword-based heuristic exhibited an accuracy of 70% whereas both NB and SVM perform equally well with an accuracy of about 79%. However, since the accuracy measure treats every class as equally important and, as we saw earlier, the dataset is imbalanced, this measure may not be best suited for comparing the different methods.

We further evaluated the classifier performance with recall and precision, which are widely used in applications where the detection of a certain class is considered more important than the detection of other classes. Recalling that our

\(^7\)http://tidy.sourceforge.net/
\(^8\)http://www.cs.waikato.ac.nz/ml/weka/
\(^9\)http://www.csie.ntu.edu.tw/~cjlin/libsvm/

<table>
<thead>
<tr>
<th>Valid APIs</th>
<th>Valid Rate</th>
<th>Invalid APIs</th>
<th>Invalid Rate</th>
<th>Total Validated APIs</th>
<th>Skipped APIs</th>
<th>Processed APIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>624</td>
<td>40.18%</td>
<td>929</td>
<td>59.82%</td>
<td>1,553</td>
<td>318</td>
<td>1,871</td>
</tr>
</tbody>
</table>

Table 1: Statistics about the validation dataset.

<table>
<thead>
<tr>
<th>Model</th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyword</td>
<td>60.3</td>
<td>75.7</td>
<td>67.0</td>
<td>70.2</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>71.0</td>
<td>79.2</td>
<td>74.8</td>
<td>78.6</td>
</tr>
<tr>
<td>SVM</td>
<td>75.4</td>
<td>70.8</td>
<td>73.1</td>
<td>79.0</td>
</tr>
</tbody>
</table>

Table 3: Classification results with different evaluation metrics.

5.1 Goal

The goal is to automatically identify Web pages describing Web APIs, the rare class, i.e., Web page relating to Web API documentation, is more valuable than the majority class, i.e., normal Web pages. Thus, we denote the former as the true class and the latter as the false class when reporting the recall and precision measure.

For each measure NB and SVM exhibited at least a 7 points increase in performance over the baseline, with the exception of the recall obtained. When comparing NB and SVM, it was observed that NB slightly outperforms SVM in terms of overall F1 score as shown in Table 3. However, inspecting the precision and recall values reveals that these two classifiers actually behave quite differently. In particular the recall of NB outperforms that of SVM in almost 9 points whereas SVM gives better prediction performance from a precision point of view for about 5 points. Considering the fact that the majority of Web pages are not related to API documentation maintaining a low level of false positive errors is important to our application, and therefore classifiers offering high precision are a more desirable choice. In this regard SVM appears as the best choice with over 15 points of performance increase compared to the baseline and 4.4 points compared to NB.

6 Conclusion and Discussion

The growing popularity of Web APIs for service-oriented software development is making more evident the need for devising technologies and systems able to better support developers in using this newer kind of services. Notably, Web APIs discovery is one aspect where substantial benefits could be obtained by using efficient automated solutions. Nowadays, ProgrammableWeb is perhaps the most popular registry for this kind of services but, although highly valuable, it presents data which is not always up to date nor directly usable by developers.

In this paper we focussed on automating the identification of Web API documentation on the Web as part of our work on a Web APIs search engine. Leveraging both direct and indirect crowdsourcing we have generated a curated dataset of URL of Web APIs that are known to provide Web API documentation. On the basis of this dataset, we have implemented an automated Web API identification engine that
Table 2: Confusion matrices for the different methods evaluated. True class corresponds to Web API documentations and false class corresponds to normal Web pages.

<table>
<thead>
<tr>
<th>Actual</th>
<th>Predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>619</td>
</tr>
<tr>
<td>True</td>
<td>153</td>
</tr>
</tbody>
</table>

(a) Keyword Heuristic

<table>
<thead>
<tr>
<th>Actual</th>
<th>Predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>785</td>
</tr>
<tr>
<td>True</td>
<td>182</td>
</tr>
</tbody>
</table>

(b) SVM

<table>
<thead>
<tr>
<th>Actual</th>
<th>Predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>727</td>
</tr>
<tr>
<td>True</td>
<td>130</td>
</tr>
</tbody>
</table>

(c) Naive Bayes

is able to properly discern Web APIs documentation with an accuracy of about 80% and a precision of 75%. Although our current implementation is solely based on document classification based on the words used, the results obtained provide a significant increase in accuracy compared to what could be obtained using directly data from ProgrammableWeb, or even compared to more advanced heuristics like the one we have used as baseline. This work represents therefore a good basis upon which we shall develop a fully-fledged Web API documentation crawler.

Additionally, this research has made more evident that the flexibility of the notion of Web API and the related technologies, although it may have had a positive impact in their adoption, it can now considerably hamper their use. In particular, our research has highlighted that due to the lack of a concrete document or Web resource that is in general used for unequivocally identifying or even describing Web APIs, their discovery, documentation and use by developers is affected. A good indicator of these difficulties is the fact that a registry like ProgrammableWeb populated mainly by service providers is not particularly well suited for finding APIs documentation able to help developers in using them.

Although the results we have obtained are promising, the tests have been realised on a small number of Web pages. In the future we plan to carry out a large-scale experiment based on the crawling of Web pages and the subsequent evaluation of the obtained results. Through these results we shall obtain a more realistic measure of the accuracy of our current approach. For this broader scale experiment we shall also incorporate additional features for the classification of documents such as the use of keywords or exploiting the structure the HTML documents.

Acknowledgments

This work was partly funded by the EU project VPH-Share (FP7-269978). The authors also would like to thank ProgrammableWeb for giving them access to their data for carrying out this research.

References


