Computing and Informatics, Vol. 40, 2021, 1345–1369, doi: 10.31577/cai_2021_6_1345

EMPIRICAL ANALYSIS ON OPENAPI TOPIC EXPLORATION AND DISCOVERY TO SUPPORT THE DEVELOPER COMMUNITY

Leonardo da Rocha Araujo, Guillermo Rodríguez
Santiago Vidal

ISISTAN-CONICET
UNICEN
Tandil, Argentina
e-mail: {leonardo.araujo, guillermo.rodriguez, santiago.vidal}@isistan.unicen.edu.ar

Claudia Marcos

ISISTAN-CIC
UNICEN
Tandil, Argentina
e-mail: claudia.marcos@isistan.unicen.edu.ar

Rodrigo Pereira dos Santos

UNIRIO
Rio de Janeiro, Brazil
e-mail: rps@uniriotec.br

Abstract. OpenAPI has become a dominant standard for documentation in the service-oriented software industry. OpenAPI is used in many analysis and reengineering approaches for RESTful service and microservice-based systems. An OpenAPI document has several components that are usually filled by humans using natural language (e.g. description of a certain functionality). Thus, subjectivity may lead to inconsistencies and ambiguities. Understanding what an API does is a challenging question. As a consequence, this issue could hinder developers from identifying the functionality of APIs, after reading all its components. Along this line, we argue that developers will be provided with supportive tools to find those APIs that better suit their needs. In this paper, we propose a step towards creating these kinds of tools by empirically analyzing a set of 2 000 OpenAPI documents with the goal of extracting the main topics of an API using three topic modeling algorithms. To address this issue, we focus on three tasks: i) determine which component of an OpenAPI document provides the most meaningful information, ii) compare three state-of-the-art topic modeling algorithms, and iii) determine the optimal number of topics to represent an API. Our findings show that the best results could be obtained from the *Description* component by using the Non-negative Matrix Factorization (NMF) or Latent Semantic Indexing (LSI) algorithms. To help developers find services in the OpenAPI directory, we also propose a prototype tool to explore the OpenAPI documents and analyze extracted topics to assess if the APIs meet developers needs.

Keywords: RESTful web services, APIs, documentation, topic modeling, OpenAPI, topic coherence

Mathematics Subject Classification 2010: 68T20, 68T50, 68U35

1 INTRODUCTION

Software developers use Application Programming Interfaces (APIs) in the software industry as a popular way to build service-oriented software applications [1]. To better understand what a specific API does, it is important to refer to its documentation. API documentation formats have been used in many analysis and re-engineering approaches for RESTful web services (also known as "lightweight" services). RESTful web services are resource-oriented services that employ the full HTTP protocol with methods like GET, POST, PUT, or DELETE as well as HTTP status codes to expose their functionality on the web [2]. In contrast to lightweight services, the "Big" Web services technology stack (related to SOAP, WSDL, WS-Addressing, WS-ReliableMessaging, and WS-Security, among others) delivers interoperability for both the Remote Procedure Call (RPC) and messaging integration styles [3].

While there are several APIs documentation standards, the most commonly used is OpenAPI[1] that focuses on describing how a specific API works. OpenAPI is based on several components. Some examples of components are the endpoints (the points where other applications might interact with) parameters that can be used, authentication methods, and other important information about that application.

Currently, there is a repository[2] with more than 2 000 APIs documented using

[1] `https://www.openapis.org/`

[2] `https://github.com/APIs-guru/openapi-directory`

OpenAPI with several components that provide information for each API. Although each OpenAPI document can be partially generated by tools, it has components (e.g. description of an endpoint) that are filled by humans. Thus, filling information is subjective and can cause inconsistencies, ambiguities and a large amount of natural language data. The manual search for the candidate APIs that meet the needs of the application may be a daunting and time-consuming task. To do so, a software developer should analyze the natural language text of the documentation for all APIs to identify the most appropriate ones. This issue could lead to a demotivating, error-prone, and tedious process [4].

A strategy to explore the identification of APIs could be the extraction of topics. These topics could not only be used later to infer the functionality of an API, but also to extract the OpenAPI' topics useful for grouping similar APIs into deployment units, such as containers [5].

In this work, we propose a first step to determine the functionality of APIs by conducting an empirical analysis of an extensive OpenAPI dataset to extract the main topics of the APIs. To achieve this goal, we used three topic modeling algorithms, namely:

1. Latent Dirichlet Allocation (LDA),
2. Latent Semantic Indexing (LSI), and
3. Non-negative Matrix Factorization (NMF).

The version of the database of OpenAPI Documentation provides 2 000 APIs. However, each API has one or more endpoints. In total there are 44 038 endpoints. Since not every endpoint has both *Summary* and *Description*, we have made an analysis only on the endpoints that have both (10 505) with the goal of making a fair comparison of the services. In summary, we are analyzing 10 505 endpoints, filtered from a total of 44 038 endpoints, that were distributed in 2 000 APIs.

Our empirical analysis consists of three tasks. A task is to determine which component of an OpenAPI document reveals the most meaningful information of an API specification. Another task is to compare the results of the three aforementioned topic modeling algorithms. Finally, we determine the optimal number of topics to represent an API. Our results show that the Description component using NMF or LSI algorithms obtain the optimal configuration in terms of topic coherence. Using these results, we build a prototype tool that provide developers with a HTML page to visualize the topics extracted from the OpenAPI dataset along with a statistical description of the results.

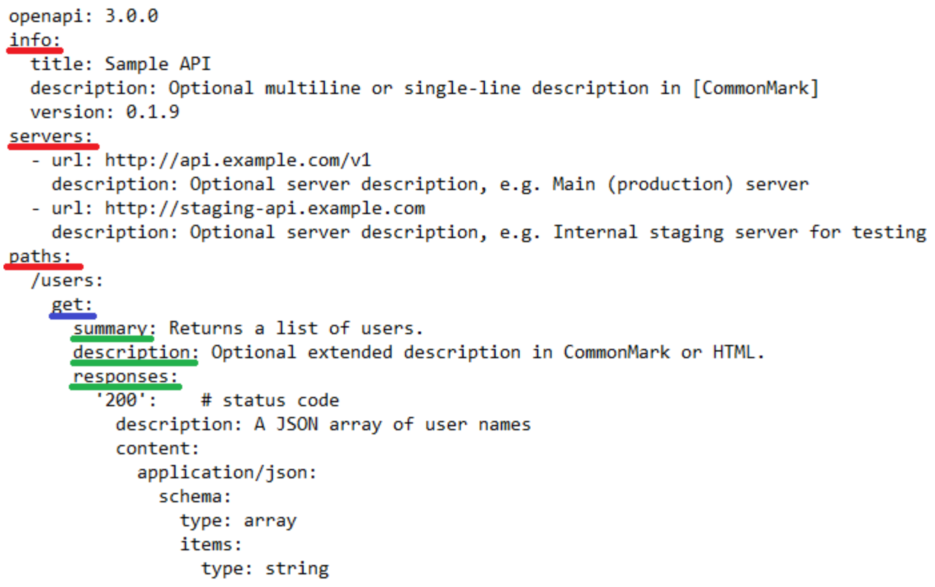
Along this line, the main contributions of this paper are:

- Measuring which OpenAPI components are more descriptive in terms of topics.
- An empirical analysis of the OpenAPI dataset in terms of topic modeling algorithms.
- A replication package with the datasets and scripts used on the experiment.

The remainder of this paper is structured as follows. Section 2 describes API documentation. Section 3 briefly presents the topic modeling algorithms used. Section 4 discusses the related work. Then, in Section 5, the design of the experiment is presented. In Section 6, the results of the experiment are reported and discussed. Section 7 presents the threats to validity of our experiment. Finally, Section 8 presents the conclusions and future work.

2 API DOCUMENTATION

There are several alternatives for API documentation. The most popular alternatives usually offer a semi-automatic tool that given the design of the application generates the structure of the documentation. Some examples are API Blueprint[3], RAML[4], and OpenAPI (previously called Swagger). All these examples are standards that can be used to design, develop, implement and document APIs. OpenAPI is presented on its website as a specification language to describe REST APIs.



```yaml
openapi: 3.0.0
info:
  title: Sample API
  description: Optional multiline or single-line description in [CommonMark]
  version: 0.1.9
servers:
  - url: http://api.example.com/v1
    description: Optional server description, e.g. Main (production) server
  - url: http://staging-api.example.com
    description: Optional server description, e.g. Internal staging server for testing
paths:
  /users:
    get:
      summary: Returns a list of users.
      description: Optional extended description in CommonMark or HTML.
      responses:
        '200':    # status code
          description: A JSON array of user names
          content:
            application/json:
              schema:
                type: array
                items:
                  type: string
```

Figure 1. Example of OpenAPI file

OpenAPI documents follow the structure shown in Figure 1. The structure is written in a yaml file. The file has 3 main components, which are underlined in red:

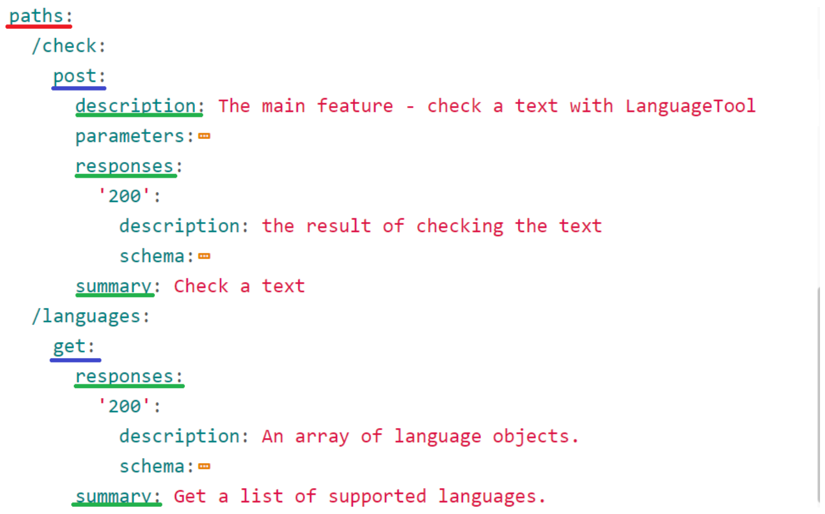
1. "Info" provides the title, version and other general information;

[3] https://apiblueprint.org/

[4] https://raml.org/

2. "Servers" has information about the servers (url, description); and
3. "Paths" defines the endpoints of the API.

The endpoint is a point of contact of any application to the API. To establish this contact, it is necessary to make a request (highlighted in blue) inside a "path" of the application. For example, the API in Figure 1 offers 1 endpoint that returns a list of users (`path: /users/get`).



```
paths:
  /check:
    post:
      description: The main feature - check a text with LanguageTool
      parameters:
      responses:
        '200':
          description: the result of checking the text
          schema:
      summary: Check a text
  /languages:
    get:
      responses:
        '200':
          description: An array of language objects.
          schema:
      summary: Get a list of supported languages.
```

Figure 2. Paths in LanguageTool API

As shown in Figure 2, a path (underlined in red) might have one or more endpoints (underlined in blue). Each endpoint has several components, but the most common are *Summary* (an endpoint's short explanation), *Description* (an endpoint's long explanation) and the possible responses of the request, all of them underlined in green.

3 TOPIC MODELING ALGORITHMS

Topic Modeling is an area of artificial intelligence that focus on finding patterns of words in document collections using hierarchical probabilistic models [26]. There are several topic modeling algorithms, but the algorithms selected were LDA, NMF and LSI. This selection stems from the fact that these are the most commonly used algorithms in topic modeling research [9, 6, 7, 20, 21, 8].

The LDA algorithm creates a set of "topics" with a probability distribution for each word to be in a topic. Then, for each text, there is a probability of each topic to be in the text, considering the words in it. The LDA algorithm only uses statistical analysis of the words and topics, therefore it does not correlate how close a word is to another to create the topics [26].

The LSI algorithm creates a vector representation for each word and checks the similarity between word vectors to create a topic. It also uses SVD (Singular Value Decomposition) to reduce a word's matrix to find synonyms. This approach allows for a topic to have a word and its synonyms with similar weights [26].

NMF algorithms approach the representation of the text and documents as numbers equal or greater than 0 (non-negative). This allows this algorithm to use mathematical formulae with this requirement to provide a faster result for queries, such as finding the texts that have a certain topic [27].

4 RELATED WORK

Several papers have been presented regarding topic modeling within applications [6, 7, 8]. Thomas et al. [8] analyzed the source code of well-designed software applications aiming to study changes in software versions. They applied topic modeling algorithms on different versions of the source code. They noticed that, for each version, the topics generated by the algorithm reflected the change in the software.

Pingclasai et al. [7] presented a bug report classification approach. They analyzed reports labeled as "bugs" or "other requests" and employed LDA to define the topics in these reports. Furthermore, they used classification algorithms (Decision Tree, Naive Bayes and Logistic Regression) with the topic modeled labeled data and have developed a model to identify bugs based on the topics in a report.

Alhindawi et al. [9] explored software documentation using topic modeling. They proposed that high-quality documentation should depict the structural organization of the source code. Therefore, they used LDA to discover topics in both the software's source code and its documentation. They used the Hellinger distance to calculate the similarity between both topics with the goal of defining how close the topics in the documentation and the source code are.

Some papers focused on Web Services discoverability and classification. Sanchez et al. [10] used WordNet, a lexical database of semantic relations between words, in Web Service Description Language (WSDL) documents to classify them by functionality. Similarly, Shafi and Qamar [11] used a text mining technique called maximum entropy to classify web services in a web service repository known as UDDI (Universal Description, Discovery and Integration).

Along this line, Mateos et al. [12] proposed a tool that uses text mining on code-first web service applications to create these documents avoiding anti-patterns caused by automatic WSDL generation. Kamath and Ananthanarayana [13] used Weka, a data mining machine learning tool, to analyze WSDL documents to find a web service that meets the user's needs.

Several studies used OpenAPI documentation as their dataset. There is some research on allowing the developer to generate code from OpenAPI documentation [1, 14, 15]. For example, Sferruzza et al. [15] used the OpenAPI standard to directly implement web services. In another work, Sferruzza et al. [14] extended the OpenAPI structure with more components that assist developers to implement top-down web services. Both studies assist developers to automatically generate the code to create web services. Baresi et al. [1] used a tool named DISCO (DIStributionally related words using CO-occurrences) to calculate semantic similarity based on the co-occurrence of these words. Then, OpenAPI's specifications are analyzed and words are mapped with DISCO.

Most of the works described in this section have utilized only a single topic modeling algorithm. However, we argue that it is important to compare the performance of the most popular algorithms in a large dataset of API's documentation. Some of the previous research that has used the OpenAPI dataset have not assessed which documentation components better describes an API document; furthermore, a comparison of information gain of those components is also important.

5 EXPERIMENT PLAN

In this section, we present our empirical evaluation of topic modeling algorithms on OpenAPI documentation following the structure proposed in [16]. The goal of this experiment is threefold:

1. finding which component of OpenAPI documents best describes the functionality of an API,
2. choosing the outperforming algorithm, and (3) determining the optimal number of topics.

To achieve the above-mentioned goal, we used the topic coherence metric [17]. This metric checks the semantic correlation between words on the same topic. It ranges from 0 to 1. Thus, a set of topics with a coherence close to 1 is most likely to have a cohesive topic structure.

The experiment aims to answer the following research questions:

RQ#1: Which OpenAPI component obtains the best results in terms of coherence for any algorithm and number of topics?

RQ#2: Which algorithm obtains the best results in terms of coherence for any number of topics and OpenAPI component?

RQ#3: Which number of topics obtains the best results in terms of coherence for any algorithm and OpenAPI component?

The directory that we are using is APIs.guru[5], which has APIs definitions in OpenAPI specification format of available documentation [18, 19]. The only restrictions for this directory are that the documentation added must be from an API, it must be written in OpenAPI specification format, and it must be publicly available. Therefore, all services that are shown in our exploration prototype are public. APIs.guru also filters non-reliable APIs, converts specifications to OpenAPI specification, and fixes mistakes on them. APIs.guru and OpenAPI themselves do not

[5] The version used in the evaluation can be found at `https://bit.ly/2yzFyAZ`

provide any information on the quality of service. Since OpenAPI is an API documentation standard, it limits itself to describe what the API does, its endpoints, response types, parameters, but not any type of quality of service metrics.

The analyzed version of "OpenAPI Directory" contained a total of 44 038 endpoints. As stated in Section 2, a functionality is an "endpoint" defined by the GET, POST, PUT, and DELETE operations inside a "path".

After analysing the dataset, we found that while some endpoints have a *Summary* and a *Description* (23.85 %), other only have one of these components or none of them (Table 1). For a fair comparison of components, we only analyzed those endpoints that have *Summary* and *Description* (= 10 505).

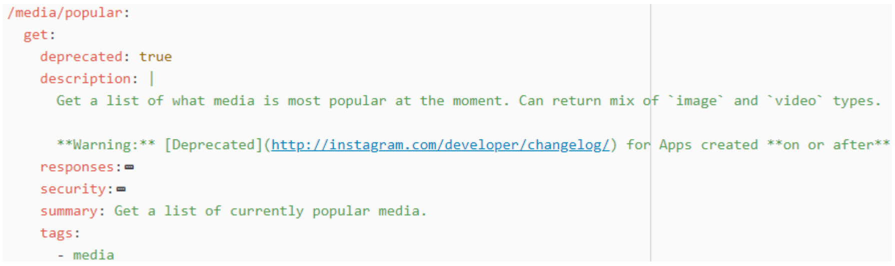
Endpoint quantity	44 038	100 %
Endpoints with *Summary*	21 470	48.75 %
Endpoints with *Description*	32 203	73.12 %
Endpoints with *Summary* and *Description*	**10 505**	**23.85 %**
Endpoints without *Summary* or *Description*	870	1.97 %
Endpoints with only *Summary*	10 965	24.90 %
Endpoints with only *Description*	21 698	49.27 %

Table 1. Number of endpoints and components

5.1 Variables

In this experiment, we focus on two components of the OpenAPI specification, namely: *Summary* and *Description*. We selected these components because they are the main sources of information in the documentation of an API which are written in natural language. *Summary* is a short text of what is done by the API, while *Description* is a long text that provides information about the API behavior. Figure 3 shows an example of an endpoint that gets a list of currently popular media from Instagram. While *Summary* only indicates that the API retrieves a list of popular media, *Description* adds information about the possible types of media (images and videos). We also analyze the combination of both components, denoted as *Summary+Description* by appending the text of the *Summary* to the *Description*. Along this line, we define these three components as the independent variables of our analysis while the topic coherence metric will be the dependant one.

Regarding the topic modeling algorithms to be compared in the experiment, we selected LDA, NMF and LSI, the most commonly used algorithms in topic modeling research [9, 6, 7, 20, 21, 8]. During the experiment, we analyze different numbers of topics ranging from 2 up to 10 topics, since it was noticed that the results are sensible to this parameter [22]. It means that these numbers of topics return coherence values that are strongly correlated with human-generated topics [22]. Thus, the degree of topic generalization is shown in several situations and, in the end, the best number of topics is chosen with the best algorithm.



```
/media/popular:
  get:
    deprecated: true
    description: |
      Get a list of what media is most popular at the moment. Can return mix of `image` and `video` types.

      **Warning:** [Deprecated](http://instagram.com/developer/changelog/) for Apps created **on or after**
    responses:▭
    security:▭
    summary: Get a list of currently popular media.
    tags:
      - media
```

Figure 3. Example of Instagram API endpoint

5.2 Hypotheses

From the research questions, we formulate three null hypotheses (the alternative hypotheses follow analogously):

- $H1_0$: The coherence value is the same for any component (*Summary*, *Description*, or *Summary* + *Description*).
- $H2_0$: The coherence value is the same for all the algorithms in a given component.
- $H3_0$: The topic coherence is the same for any number of topics.

5.3 Design

With the goal of answering the research questions, we considered three different scenarios to run the algorithms:

- using only the *Summary* information,
- using only the *Description* information,
- using *Summary* and *Description* information.

In each scenario, we train a model that defines from 2 to 10 topics by using LDA, LSI and NMF. In the end, 81 models (9 topics $\times$ 3 algorithms $\times$ 3 components) are evaluated using the topic coherence metric to define the configuration that answers our research questions.

5.4 Procedure

The experiment consisted of four phases[6], namely

1. Extract information,
2. Preprocess text,

[6] The code is hosted at `https://github.com/OpenAPIAnalysis/OAPIAnalysis`

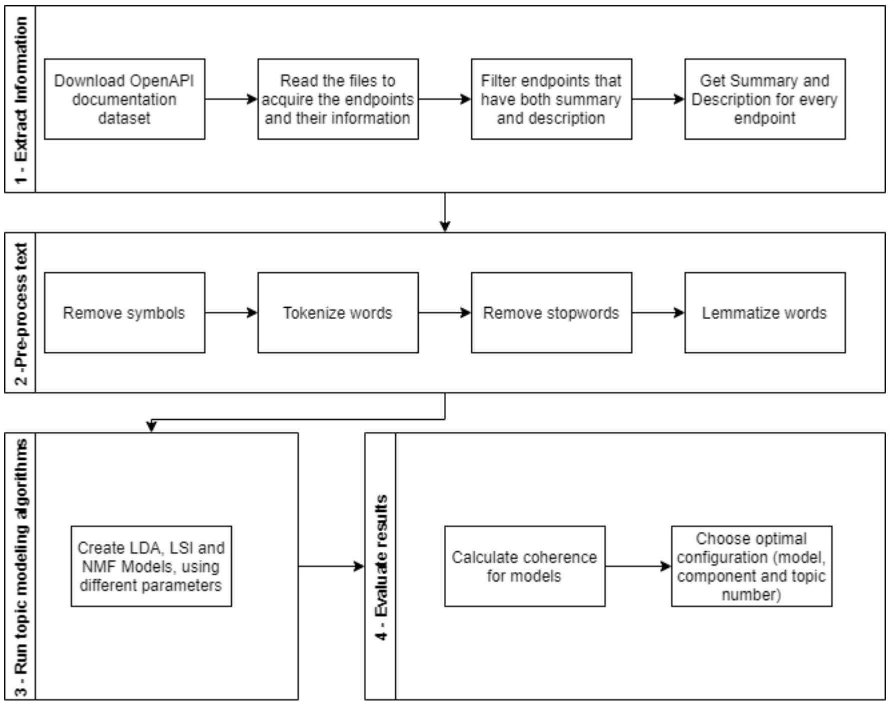

1 - Extract Information
Download OpenAPI documentation dataset
Read the files to acquire the endpoints and their information
Filter endpoints that have both summary and description
Get Summary and Description for every endpoint
2 -Pre-process text
Remove symbols
Tokenize words
Remove stopwords
Lemmatize words
3 - Run topic modeling algorithms
Create LDA, LSI and NMF Models, using different parameters
4 - Evaluate results
Calculate coherence for models
Choose optimal configuration (model, component and topic number)

Figure 4. Phases of the experiment

3. Run topic modeling algorithms, and
4. Evaluate results.

Figure 4 shows the four phases as groups of actions with each action summarized in a box, these actions are explained in the following paragraphs.

1. **Extract information.** On the first phase, we have downloaded the OpenAPI documentation dataset. Then, each documentation file was read and the information on their endpoints was stored. The endpoints with both *Summary* and *Description* were filtered, to avoid bias in the vocabulary. To extract the *Summary* and *Description* texts for each endpoint, we have stored them by defining 3 variables: *summary*, *description*, and "*summary_and_description*".
2. **Preprocess text.** We preprocessed the text stored in the database by removing symbols in the text, applying tokenization, removing stopwords and applying lemmatization [23], as shown in the second phase of the Figure 4. Tokenization consists of converting words to separated words (called tokens). Stopwords removal consists of discarding words that often appear in the vocabulary and fails to add meaning. Lemmatization consists of reducing a word to a common root.

3. **Run topic modeling algorithms.** The corpus resulting from the preprocessing was given as input to the LDA, LSI and NMF algorithms, all implemented in Python's library Gensim. The algorithms generated a model for 2 to 10 topics, with each topic being a set of words with a weight on each of them. The weight is determined by the topic modeling algorithm. The experiment was run setting the random seed to 1, so the results would be reproducible.
4. **Evaluate results.** To evaluate the results, we used the topic modeling coherence model, which focuses on finding the semantic correlation between a number of words with the highest weight in a topic (referred in Gensim's algorithm as Top N words). The topic modeling coherence model has demonstrated that it defines topics similar to topics assigned by humans [17, 24].

However, Röder et al. [24] have calculated coherence using six values, top 5 up to 10 words, since more words might generate noise; besides, they have shown that this number of words is enough to evaluate the topic. Additionally, the same authors have shown that values that indicate a strong correlation are between the values of 0.6 to 0.8. Below this value, the correlation is considered weak, while above this value the model is probably overfitted.

The coherence values were calculated for all models. A simplified pseudocode of the process is illustrated in Listing 1. The process consists of iterating over the number of topics to be used in the topic modeling (using the variable *topic_qtt*), the three different algorithms to be used (using the variable *algorithm*) and the possible values for the top words (using the variable *topn*) and calculating the coherence. Given each configuration and the corpus of text previously created, the model is set and stored in the variable *model*. Finally, the coherence of the model is calculated, in relation to the corpus, according to the value set in *topn*. As mentioned before, the values used in this calculation are the top 5 words up to the top 10 words. Therefore, we have created models for 3 different topic modeling algorithms (LDA, LSI, NMF), for 3 different components (*Summary*, *Description* and *Summary + Description*), for 9 different topic quantities (2 up to 10), and which gave us in total 81 models. For each model, 6 coherence values (top 5 words up to top 10 words) were calculated, which gave us in total 486 coherence measurements. Lastly, the model with the optimal configuration is chosen.

```
foreach number of topic quantity:
  for each of the topic modeling algorithms:
    for each number of topn:
      if algorithm is LDA:
        model = gensim.models.LdaModel(corpus, topic_qtt)
      end if
      if algorithm is LSI:
        model = gensim.models.LsiModel(corpus, topic_qtt)
      end if
      if algorithm is NMF:
        model = gensim.models.NMFModel(corpus, topic_qtt)
      end if
      c_v = CoherenceModel(model, corpus, topn)
    endfor
  endfor
endfor
```

Listing 1. Simplified pseudocode of the evaluation process

6 RESULTS

This section presents the results[7] of our experiments and answers to the research questions.

6.1 RQ#1: Which OpenAPI Component Obtains the Best Results in Terms of Coherence for Any Algorithm and Number of Topics?

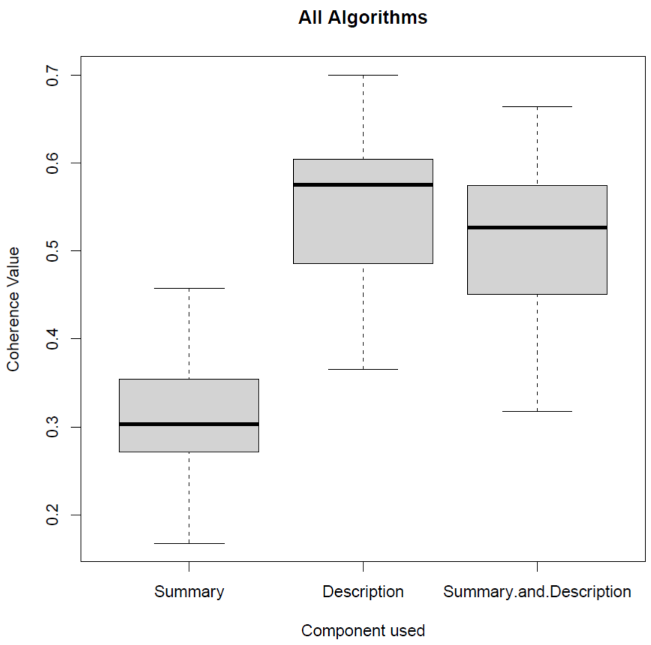


Figure 5. Coherence values by component

Figure 5 plots the results. Results show that the scenario in which only *Summary* was considered has the worst coherence value in every algorithm compared to the

[7] All the results can be accessed at `https://bit.ly/3f7G2yH`

other components. In fact, the highest value for only considering *Summary* was 0.45 and was obtained by the LSI (Figure 6). While the lowest value was obtained by the same algorithm and was 0.16. In the case of only considering *Description* (Figure 7), the highest value obtained was 0.65 for the LSI algorithm and the lowest value, 0.54 (also for the same algorithm). Moreover, the average is above *Summary*'s average in every value. When the component *Summary* + *Description* is considered, the higher value of 0.64 was obtained for the NMF algorithm (Figure 8).

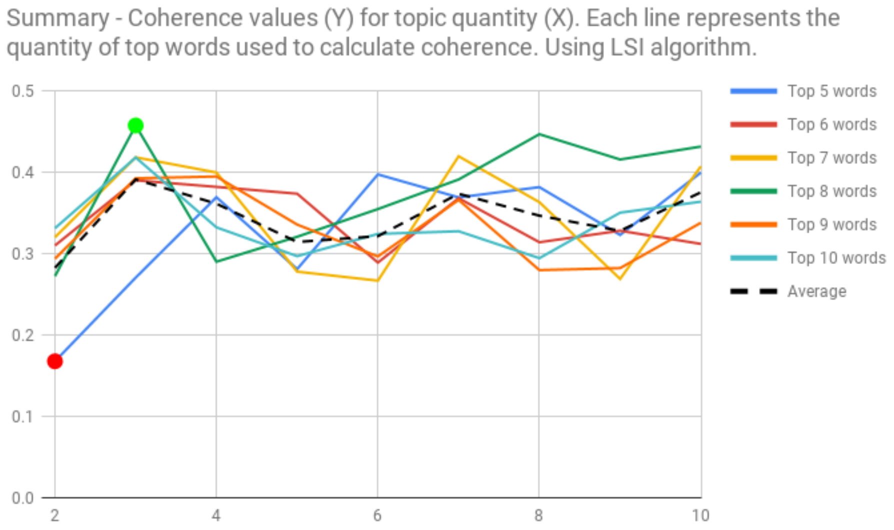


Figure 6. LSI algorithm applied to summaries

One of the reasons that might justify *Summary*'s low coherence values is the fact that it has the smallest vocabulary and small sentences that repeat keywords often. This might make it harder for the algorithm to extract relevant information. This also explain why *Description*'s coherence drops when *Summary* is appended to it, since *Summary* generates noise to *Description*. Since *Description*'s vocabulary is significantly larger than *Summary*'s and their sentences have a higher diversity of words, the coherence drops slightly. Also, although *Summary* + *Description* has a coherence value close to the *Description*'s, the coherence drops because of the noise generated by the *Summary* appended to *Description*. It is possible to use either *Description* or *Summary* + *Description*, since both coherence values are above the threshold to indicate a correlation between topics. Using *Summary* as the component to model topics is unlikely to generate coherent topics, since its coherence values are below 0.6, the threshold to indicate a correlation.

Although these values give descriptive insights, no conclusion can be made so far on whether there is a significant difference between the components. A statistical

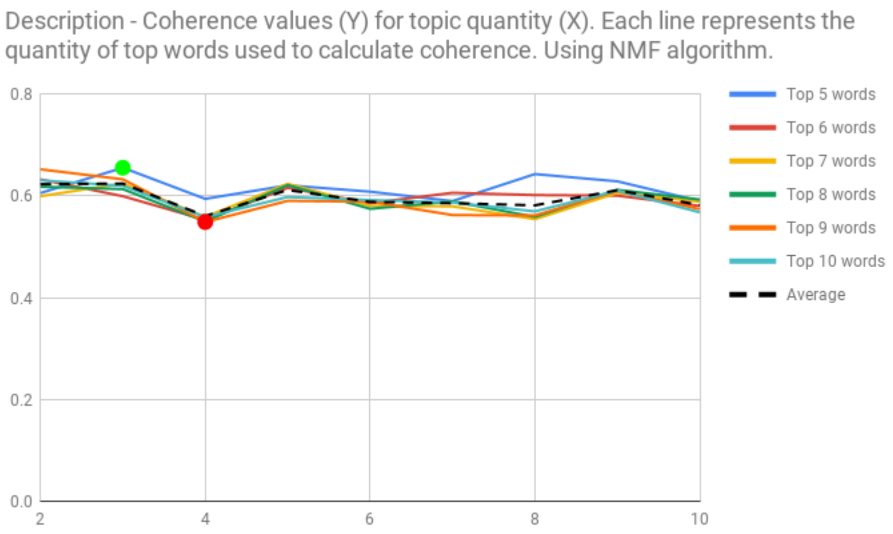


Figure 7. LSI algorithm applied to descriptions

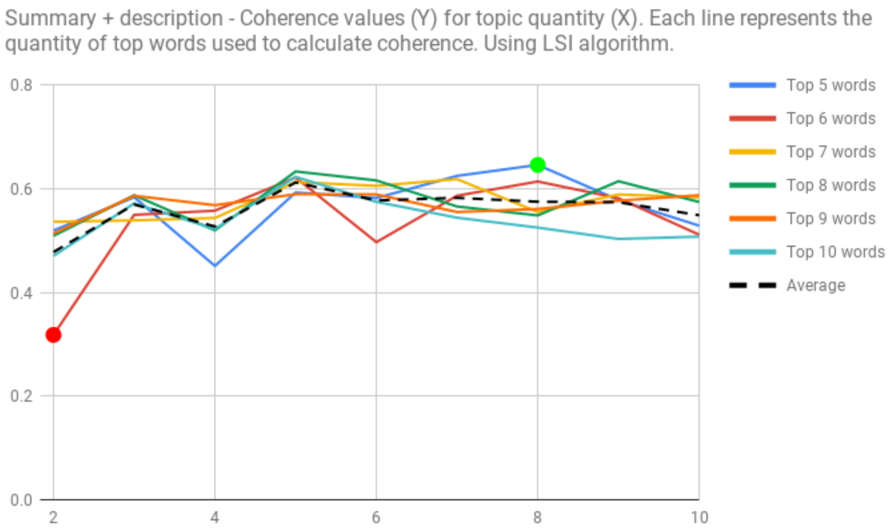


Figure 8. NMF algorithm applied to Summaries + Descriptions

test is necessary. Our hypothesis $H1_0$ stated that the coherence value is the same for any component. To evaluate that, we used the Analysis of Variance (ANOVA). ANOVA is a parametric statistical test used with the objective of testing the statistical significance of mean differences in different groups of scores [25]. It is calculated as the ratio of the variance between several groups of data and the variance within the groups.

	ANOVA	
	F-statistic	p-value
$H1_0$	509.6155	< 0.0001
$H2_0$	46.0913	< 0.0001
$H3_0$	0.8446186589	0.5621

Table 2. ANOVA results

After running the test, we were able to reject $H1_0$ with a p-value < 0.0001 (Table 2). Thus, there is statistical significance difference between the coherence values obtained for the components.

To determine which pairs of the components are statistically different, a post-hoc analysis is needed. We applied the Scheffé test [25] which makes a pair-wise comparisons between the groups of data. After running the test, we found that there is a significant difference between all the comparisons of groups (Table 3).

	Scheffe		
	Hypothesis 1		
	T-Statistic	p-value	Inference
Sum $- S + D$	25.3441	1.11E$-$16	significant
Des $- S + D$	4.1411	2.19E$-$04	significant

Table 3. Table for Scheffé test – $H1$

Therefore, to answer to RQ#1, the component that obtains the best results in terms of coherence for any algorithm is the Description.

6.2 RQ#2: Which Algorithm Obtains the Best Results in Terms of Coherence for Any Number of Topics and OpenAPI Component?

To answer this question, a boxplot was made with the coherence values grouped by algorithm (Figure 9). The boxplot shows that most of the values in NMF and LSI algorithm are higher than LDA, and the mean of both also is higher, therefore both algorithms are candidates for optimal performance.

In order to claim any significant difference, a statistical test must be applied. Our hypothesis $H2_0$ stated that the coherence value is the same for all the algorithms in a given component. Table 2 shows that there is a significant difference between algorithms, since the p-value is below 0.05. After executing the Scheffé test, it was

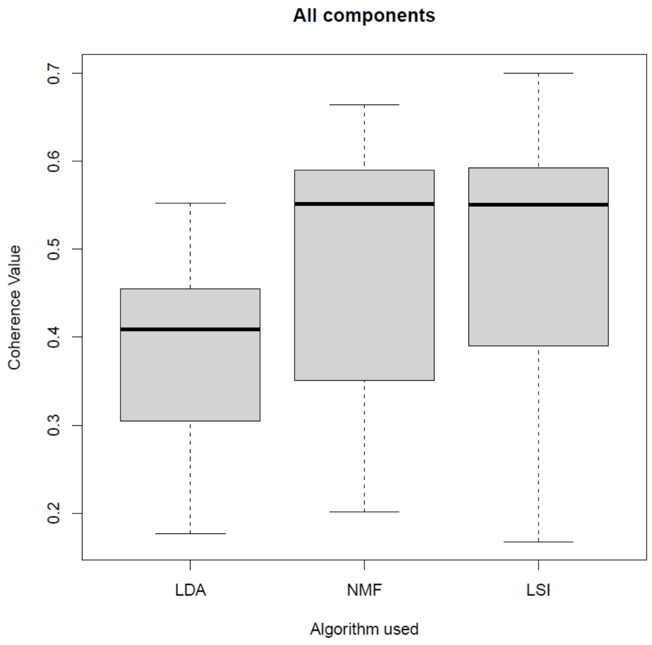


Figure 9. Coherence values by algorithm

possible to detect that there is a significant difference between the algorithms LDA and the other two algorithms, but there is not a significant difference between NMF and LSI algorithms (Table 4).

Since in RQ#1 we found that higher coherence values were obtained by using the component Description, we also analyzed the boxplot by only taking into account that component (Figure 10). We found similar results than when all the components were analyzed: lower coherence values for LDA compared to NMF and LSI and no significant difference between NMF and LSI.

	Scheffe Hypothesis 2		
	T-Statistic	p-value	Inference
LDA – NMF	7.7767	4.24E−13	significant
LDA – LSI	8.7648	3.33E−16	significant
NMF – LSI	0.9881	6.14E−01	not significant

Table 4. Table for Scheffé test – $H2$

Therefore, to answer to RQ#2, the algorithms that obtain the optimal performance in terms of coherence for any number of topics are LSI and NMF.

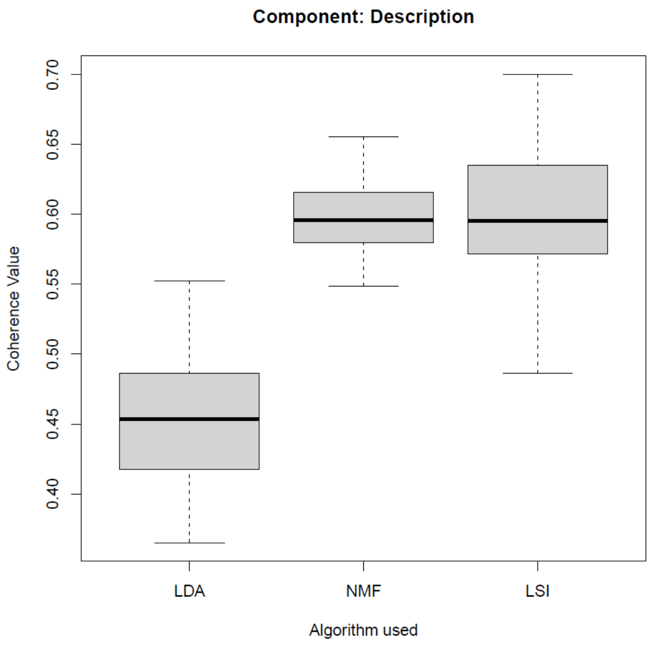


Figure 10. Coherence values by algorithm for description

6.3 RQ#3: Which Number of Topics Obtains the Best Results in Terms of Coherence for Any Algorithm and OpenAPI Component?

In order to answer to this question, we analyzed the coherence values resulting from running the algorithms with different number of topics. As stated in Section 5.1, we run the algorithms with topics ranging from 2 to 10. Figure 11 plots our results. As it is shown, most of the medians are between 0.4 and 0.5.

In order to claim any statistical difference, we tested our results with ANOVA.

As seen in Table 2, the p-value calculated exceeds the threshold of 0.05. Therefore, there is no significant difference between the coherence values obtained by the different topic numbers.

Since in RQ#1 we found that higher coherence values were obtained by using the component Description, and in RQ#2 we found that the best algorithms are LSI and NMF, we also analyzed the results taking into consideration only the coherence values obtained with these variables. Figure 12 plots these results. As it can be seen, the medians of the coherence values increased notably to the range of 0.55 to 0.6. After running the ANOVA test for these results, we were able to reject $H3_0$ with a p-value = 0.0198. However, after running the post-hoc test, we only found a statistically significant difference between the results obtained with 2 and 7 topics.

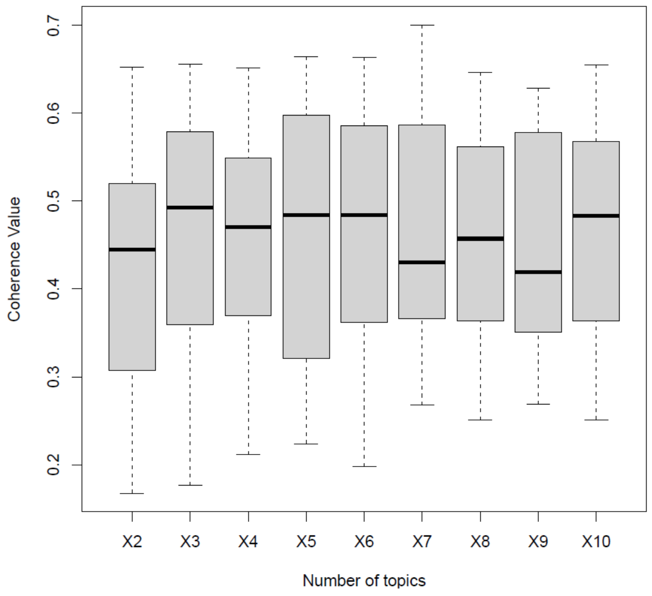


Figure 11. Boxplot representing coherence values by number of topics

Therefore, to answer to RQ#3, we can state that it was not possible to find an optimal number of topics. However, we found a significant difference in the coherence values obtained when only the *Description* component and the LSI and NMF algorithms were used.

6.4 A Prototype to Explore OpenAPI Documents

The results achieved determine configurations that generate the models with the highest coherence possible. The usage of these models varies depending on the objective. Our objective is to evaluate the possible configurations and propose the best option to be used to help the developers from the API Community to explore services in OpenAPI Dataset. To help developers to find services in the OpenAPI directory, we also propose an exploration tool that separates the endpoints in the topics that are more related to them. To illustrate this usage, we have developed a script that does a top-level exploration.

Our script evaluates the models generated by the files downloaded in OpenAPI Directory, providing the configuration with the highest topic coherence. Once this is done, we also provide a top-level visualization of the results for each topic. Our script generates an HTML file with 2 tables, presenting information about the 12 topics modelled. On the first table, the 5 endpoints with best score for each topic are shown, alongside the topic number, their coherence and the words that have the

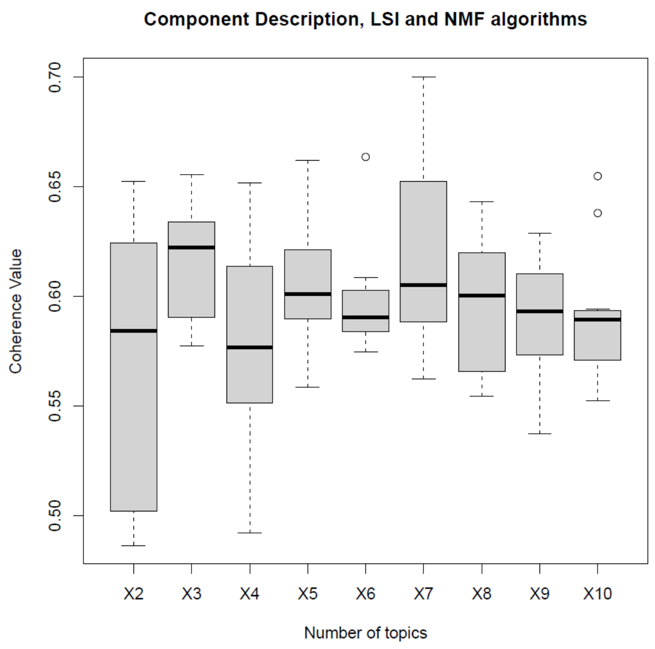
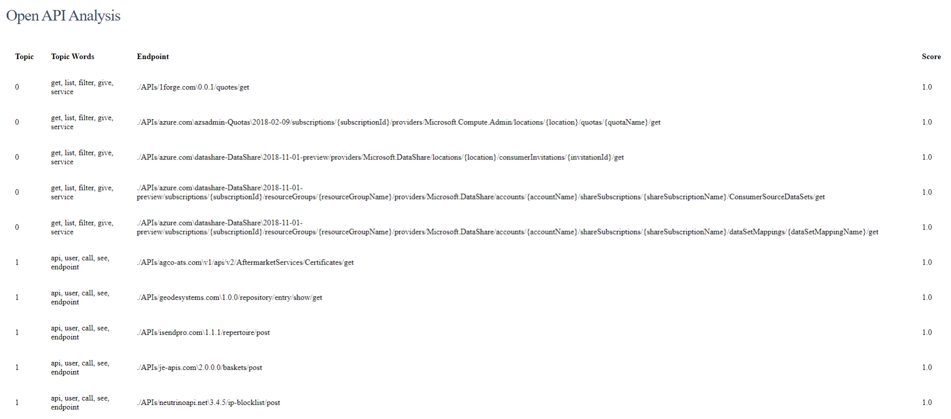


Figure 12. Boxplot representing coherence values for LSI and NFM algorithms for Description component by number of topics

highest weight on the topic (most relevant words), while the other endpoints are omitted. This is done to inform the 5 endpoints that are most related to each topic. On the second table, it is shown the amount of endpoints in each topic, the coherence of each topic and other statistical information.

The coherence value for each topic, which determines how correlated are the top words in the topics. The average score (and standard deviation), which shows the average of the scores, basically a numeric value to determine how aligned the endpoints are to the topics, whether the standard deviation defines the amount of variation in the values. The average weight of the top 5 words (and standard deviation), which measures how important (concerning other topics) the top 5 words in a topic is, and its standard deviation shows the variance in these values. Lastly, the quantity of endpoints in each topic, which shows how populated each topic is, therefore providing information on the most relevant topics for the whole dataset. The first table helps the developers to find services related to each topic, and the second table helps to understand how much the services in each topic represent each topic.

Our script is a prototype of a tool that can be used to help developers find the services they need. Nowadays if a user needs to find a service in the OpenAPI directory, they need to manually check each service to make sure that they did not

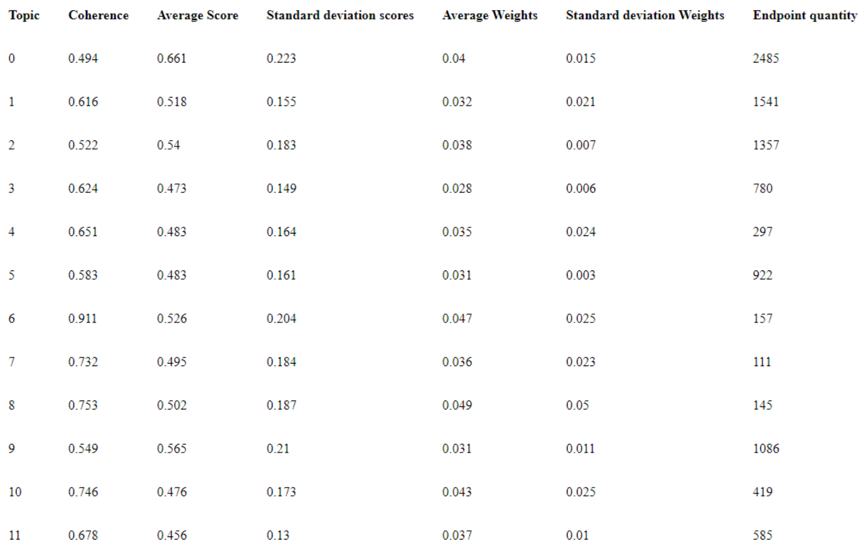


Open API Analysis

Topic	Topic Words	Endpoint	Score
0	get, list, filter, give, service	./APIs/1forge.com\0.0.1/quotes/get	1.0
0	get, list, filter, give, service	./APIs/azure.com\azsadmin-Quotas\2018-02-09/subscriptions/{subscriptionId}/providers/Microsoft.Compute.Admin/locations/{location}/quotas/{quotaName}/get	1.0
0	get, list, filter, give, service	./APIs/azure.com\datashare-DataShare\2018-11-01-preview/providers/Microsoft.DataShare/locations/{location}/consumerInvitations/{invitationId}/get	1.0
0	get, list, filter, give, service	./APIs/azure.com\datashare-DataShare\2018-11-01-preview/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.DataShare/accounts/{accountName}/shareSubscriptions/{shareSubscriptionName}/ConsumerSourceDataSets/get	1.0
0	get, list, filter, give, service	./APIs/azure.com\datashare-DataShare\2018-11-01-preview/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.DataShare/accounts/{accountName}/shareSubscriptions/{shareSubscriptionName}/dataSetMappings/{dataSetMappingName}/get	1.0
1	api, user, call, see, endpoint	./APIs/agco-ats.com\v1/api/v2/AftermarketServices/Certificates/get	1.0
1	api, user, call, see, endpoint	./APIs/geodesystems.com\1.0.0/repository/entry/show/get	1.0
1	api, user, call, see, endpoint	./APIs/isendpro.com\1.1.1/repertoire/post	1.0
1	api, user, call, see, endpoint	./APIs/je-apis.com\2.0.0.0/baskets/post	1.0
1	api, user, call, see, endpoint	./APIs/neutrinoapi.net\3.4.5/ip-blocklist/post	1.0

Figure 13. Excerpt of first table shown in the HTML file, presenting a top level exploration of the topics and the endpoints most related to these topics

miss what they need. An exploration tool helps the developers to execute this task more effectively and avoid errors. Therefore, in future work, we can improve this tool to enhance the exploration aspect. One example of enhancement of the tool is to use the topic modeling in each topic, recursively, to generate sub-topics for each topic.



Topic	Coherence	Average Score	Standard deviation scores	Average Weights	Standard deviation Weights	Endpoint quantity
0	0.494	0.661	0.223	0.04	0.015	2485
1	0.616	0.518	0.155	0.032	0.021	1541
2	0.522	0.54	0.183	0.038	0.007	1357
3	0.624	0.473	0.149	0.028	0.006	780
4	0.651	0.483	0.164	0.035	0.024	297
5	0.583	0.483	0.161	0.031	0.003	922
6	0.911	0.526	0.204	0.047	0.025	157
7	0.732	0.495	0.184	0.036	0.023	111
8	0.753	0.502	0.187	0.049	0.05	145
9	0.549	0.565	0.21	0.031	0.011	1086
10	0.746	0.476	0.173	0.043	0.025	419
11	0.678	0.456	0.13	0.037	0.01	585

Figure 14. Excerpt of second table shown in the HTML file, presenting a statistical analysis of the topics and endpoints related to them

7 THREATS TO VALIDITY

The threats to validity are divided into 4 categories: conclusion, internal, construct and external [16].

In conclusion validity, it is analyzed the ability to draw inaccurate results from the observations in the experiment. To avoid threats in this type of validity we used previously tested metrics to reach the results of our models and then analyzed them with statistical methods, such as ANOVA and post-hoc tests. Also, a relatively large dataset was used, with over 10 000 endpoints, to generate the models, providing models with data variation. Our original dataset had more endpoints to be used, but some of them only had information in one component. To avoid another threat to the conclusion validity we used only endpoints with information in both components, so the data would not have a bias towards the component with more endpoints.

The internal validity is defined by the causal relations in the experiment. One of the threats we could not avoid is that the text we use in our dataset is made by humans, subjectively. Therefore, it might have orthography errors or it might not be an accurate description of the respective endpoint.

Construct validity guarantees that the experiment done can generalize the theory behind it. To minimize this type of threat, we chose components that have the function to directly describe the object we are analyzing to obtain its characteristics. We could not test if the results proposed by our study model the characteristics, because this would require manual tests with researchers, which is not the scope of this project. Therefore, we proposed it as future work.

The external validity is concerned with generalizing the results to other environments. To reduce these threats, we used in our experiment a large database containing APIs from different domains. Since the repository used was APIs Guru, there was a limitation of only using public APIs, which were made with a long-lived goal (persistent). Therefore, it is possible that certain vocabulary from private API is missing, such as bank services. Alternatively, this repository also filters non-reliable API and fixes mistakes in the definitions, which increases the internal validity.

8 CONCLUSION AND FUTURE WORK

In this paper, we presented an empirical analysis of an extensive OpenAPI dataset to extract the main topics of APIs. OpenAPI documentation contains numerous components to provide information about an API, for example a *Summary* and a *Description* of the endpoint. In this paper, we examined which component of the OpenAPI documentation might provide better topics in terms of the coherence metric. Such topics could later be used to gather an API's functionality.

To conduct our empirical analysis, we used three topic modeling algorithms in a dataset of over 2 000 OpenAPI documents, namely LDA, LSI, and NMF. Backed up with statistical results, we found that in terms of topic coherence the component Description of an OpenAPI document best describes the functionality of an API by using the NMF or LSI algorithms. Also, we found that the coherence metric values are stable for different number of topics. Furthermore, we have built a prototype tool to assist developers in exploring OpenAPI documents and analyzing extracted topics to assess if the APIs meet developers needs.

As future work, we intend to corroborate if topics can effectively be used to determine the functionality of an API. Also, we plan to take advantage of the knowledge gained from this experiment and explore unsupervised approaches that use pre-trained word/sentence embeddings to cluster similar APIs from a functional viewpoint. Another important issue is to use the model, created in this paper, to compare against the characteristics created by researchers manually to test how well our model is defining characteristics of APIs compared to researchers. Finally, we intend to analyze other OpenAPI repositories with a higher number of OpenAPI documents and focus on those documents that do not specify web APIs, such as microservices that are just made to run in a cluster and without exposing themselves outside (to the Internet).

Acknowledgement

We acknowledge the financial support provided by ANPCyT (Argentina) under the Project PICT 2018-01456 and by CONICET (Argentina) under a doctoral grant and PIP 2021-2023 id 11220200100430CO. The fifth author thanks to UNIRIO and FAPERJ (Proc. 211.583/2019) for their partial support.

REFERENCES

[1] Baresi, L.—Garriga, M.—De Renzis, A.: Microservices Identification Through Interface Analysis. In: De Paoli, F., Schulte, S., Broch Johnsen, E. (Eds.): Service-Oriented and Cloud Computing (ESOCC 2017). Springer, Cham, Lecture Notes in Computer Science, Vol. 10465, 2017, pp. 19–33, doi: 10.1007/978-3-319-67262-5_2.

[2] Bogner, J.—Wagner, S.—Zimmermann, A.: Collecting Service-Based Maintainability Metrics from RESTful API Descriptions: Static Analysis and Threshold Derivation. In: Muccini, H. et al. (Eds.): Software Architecture (ECSA 2020). Springer, Cham, Communications in Computer and Information Science, Vol. 1269, 2020, pp. 215–227, doi: 10.1007/978-3-030-59155-7_16.

[3] Pautasso, C.—Zimmermann, O.—Leymann, F.: Restful Web Services vs. "Big" Web Services: Making the Right Architectural Decision. Proceedings of the 17th International Conference on World Wide Web (WWW '08), 2008, pp. 805–814, doi: 10.1145/1367497.1367606.

[4] Tyszberowicz, S.—Heinrich, R.—Liu, B.—Liu, Z.: Identifying Microservices Using Functional Decomposition. In: Feng, X., Müller-Olm, M., Yang, Z. (Eds.): Dependable Software Engineering. Theories, Tools, and Applications (SETTA 2018). Springer, Cham, Lecture Notes in Computer Science, Vol. 10998, 2018, pp. 50–65, doi: 10.1007/978-3-319-99933-3_4.

[5] Lin, M.—Xi, J.—Bai, W.—Wu, J.: Ant Colony Algorithm for Multi-Objective Optimization of Container-Based Microservice Scheduling in Cloud. IEEE Access, Vol. 7, 2019, pp. 83088–83100, doi: 10.1109/ACCESS.2019.2924414.

[6] Nguyen, A. T.—Nguyen, T. T.—Nguyen, T. N.—Lo, D.—Sun, C.: Duplicate Bug Report Detection with a Combination of Information Retrieval and Topic Modeling. Proceedings of the 2012 27th IEEE/ACM International Conference on Automated Software Engineering, 2012, pp. 70–79, doi: 10.1145/2351676.2351687.

[7] Pingclasai, N.—Hata, H.—Matsumoto, K. I.: Classifying Bug Reports to Bugs and Other Requests Using Topic Modeling. 2013 20th Asia-Pacific Software Engineering Conference (APSEC), IEEE, Vol. 2, 2013, pp. 13–18, doi: 10.1109/APSEC.2013.105.

[8] Thomas, S. W.—Adams, B.—Hassan, A. E.—Blostein, D.: Studying Software Evolution Using Topic Models. Science of Computer Programming, Vol. 80, 2014, Part B, pp. 457–479, doi: 10.1016/j.scico.2012.08.003.

[9] Alhindawi, N.—Al-Hazaimeh, O. M.—Malkawi, R.—Alsakran, J.: A Topic Modeling Based Solution for Confirming Software Documentation Quality. International Journal of Advanced Computer Science and Applications, Vol. 7, 2016, No. 2, pp. 200–206, doi: 10.14569/IJACSA.2016.070227.

[10] Sánchez-Sánchez, C.—Villatoro-Tello, E.—Ramírez-de-la-Rosa, G.—Jiménez-Salazar, H.—Pinto, D.: WSDL Information Selection for Improving Web Service Classification. Research in Computing Science, Vol. 144, 2017, No. 1, pp. 83–96, doi: 10.13053/rcs-144-1-7.

[11] Shafi, S.—Qamar, U.: [WiP] Web Services Classification Using an Improved Text Mining Technique. 2018 IEEE 11th Conference on Service-Oriented Computing and Applications (SOCA), IEEE, 2018, pp. 210–215, doi: 10.1109/SOCA.2018.00037.

[12] Mateos, C.—Rodriguez, J. M.—Zunino, A.: A Tool to Improve Code-First Web Services Discoverability Through Text Mining Techniques. Software: Practice and Experience, Vol. 45, 2015, No. 7, pp. 925–948, doi: 10.1002/spe.2268.

[13] Kamath, S. S.—Ananthanarayana, V. S.: Semantic Similarity Based Context-Aware Web Service Discovery Using NLP Techniques. Journal of Web Engineering, Vol. 15, 2016, No. 1-2, pp. 110–129.

[14] Sferruzza, D.: Top-Down Model-Driven Engineering of Web Services from Extended OpenAPI Models. 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE), 2018, pp. 940–943, doi: 10.1145/3238147.3241536.

[15] Sferruzza, D.—Rocheteau, J.—Attiogbé, C.—Lanoix, A.: Extending OpenAPI 3.0 to Build Web Services from Their Specification. Proceedings of the 14th International Conference on Web Information Systems and Technologies (APMDWE), Vol. 1, 2018, pp. 412–419, doi: 10.5220/0006923604120419.

[16] Wohlin, C.—Runeson, P.—Höst, M.—Ohlsson, M. C.—Regnell, B.—Wesslén, A.: Experimentation in Software Engineering. Springer Science and Business Media, 2012, doi: 10.1007/978-3-642-29044-2.

[17] Chang, J.—Gerrish, S.—Wang, C.—Boyd-Graber, J.—Blei, D. M.: Reading Tea Leaves: How Humans Interpret Topic Models. In: Bengio, Y., Schuur-

mans, D., Lafferty, J., Williams, C., Culotta, A. (Eds.): Advances in Neural Information Processing Systems 22 (NIPS 2009), 2009, pp. 288–296.

[18] Koren, I.—Klamma, R.: The Exploitation of OpenAPI Documentation for the Generation of Web Frontends. Companion Proceedings of The Web Conference 2018 (WWW '18), 2018, pp. 781–787, doi: 10.1145/3184558.3188740.

[19] Yasmin, J.—Tian, Y.—Yang, J.: A First Look at the Deprecation of RESTful APIs: An Empirical Study. 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2020, pp. 151–161, doi: 10.1109/ICSME46990.2020.00024.

[20] Řehůřek, R.—Sojka, P.: Software Framework for Topic Modelling with Large Corpora. Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, 2010, pp. 46–50, doi: 10.13140/2.1.2393.1847.

[21] Thiele, T.—Sommer, T.—Stiehm, S.—Jeschke, S.—Richert, A.: Exploring Research Networks with Data Science: A Data-Driven Microservice Architecture for Synergy Detection. 2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW), 2016, pp. 246–251, doi: 10.1109/W-FiCloud.2016.58.

[22] Grant, S.—Cordy, J. R.—Skillicorn, D. B.: Using Heuristics to Estimate an Appropriate Number of Latent Topics in Source Code Analysis. Science of Computer Programming, Vol. 78, 2013, No. 9, pp. 1663–1678, doi: 10.1016/j.scico.2013.03.015.

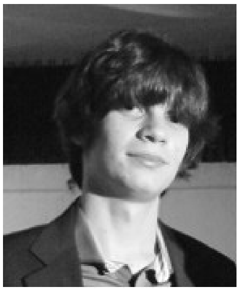
[23] Feldman, R.—Sanger, J.: The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data. Cambridge University Press, 2006, doi: 10.1017/CBO9780511546914.

[24] Röder, M.—Both, A.—Hinneburg, A.: Exploring the Space of Topic Coherence Measures. Proceedings of the Eighth ACM International Conference on Web Search and Data Mining (WSDM '15), 2015, pp. 399–408, doi: 10.1145/2684822.2685324.

[25] Tabachnick, B. G.—Fidell, L. S.: Experimental Designs Using ANOVA. Second Edition. Thomson/Brooks/Cole, Belmont, CA, 2007.

[26] Alghamdi, R.—Alfalqi, K.: A Survey of Topic Modeling in Text Mining. International Journal of Advanced Computer Science and Applications, Vol. 6, 2015, No. 1, pp. 147–153, doi: 10.14569/IJACSA.2015.060121.

[27] Pauca, V. P.—Piper, J.—Plemmons, R. J.: Nonnegative Matrix Factorization for Spectral Data Analysis. Linear Algebra and Its Applications, Vol. 416, 2006, No. 1, pp. 29–47, doi: 10.1016/j.laa.2005.06.025.



Leonardo da Rocha Araujo is Ph.D. student at the UNICEN University. His primary research interests are microservice architecture and natural language processing.

Guillermo Rodríguez is currently a member of the ISISTAN Research Institute (CONICET-UNICEN), Adjunct Researcher at the CONICET and Professor at the UNCPBA. He graduated as Systems Engineer in 2010 (UNCPBA), and Doctor of Computer Science in 2014 (UNCPBA). His main research areas are service oriented software development and case based reasoning.

Santiago Vidal is Professor at the UNICEN University, and also Research Fellow of the CONICET, Argentina. His primary research interests are software evolution and maintenance. He received his M.Sc. and Ph.D. degrees in computer science from the UNICEN University in 2011 and 2013, respectively.

Claudia Marcos has been Professor in the School of Computer Science at Universidad Nacional del Centro de la Provincia de Buenos Aires (UNCPBA) since 1991. She is CIC (Comisión de Investigación Científica de la Provincia de Buenos Aires) Researcher. Her main research area is in software evolution, requirements engineering and agile development. She teaches several undergraduate and postgraduate courses at the UNICEN and has also national and international publications in the area. She received her B.Sc. degree in 1993 from the UNCPBA State University in 1993. She obtained her Ph.D. degree in computer science in 2001.

Rodrigo Pereira dos Santos received his Ph.D. degree in computer science from the Alberto Luiz Coimbra Institute for Graduate Studies and Research in Engineering, Federal University of Rio de Janeiro (UFRJ), 2016. He is currently Associate Professor with the Department of Applied Informatics, Federal University of the State of Rio de Janeiro (UNIRIO), where he leads the Complex Systems Engineering Laboratory (LabESC). His research interests include complex systems engineering (specially software ecosystems and systems-of-systems) and software engineering education.