Augmenting API documentation with insights from stack overflow [Replication]

Ovietobore Oghre
Department of Computer Science
University of Manitoba
Winnipeg, MB, CA
oghreo@myumanitoba.ca

***Abstract*—Software documentation is extremely important to software developers. However, there are often problems with software documentations. For example, sometimes documentation is incomplete or the documentation tells developers what to do, but does not offer any reasons why. To attempt to solve this problem, in this paper, I will duplicate an approach to automatically augment API documentation with insight sentences (sentences that are not contained in the API documentation and add additional information not included in the documentation) from Stack Overflow. This project uses a machine learning approach using stack overflow meta data along with data from the sentences themselves as features. For this project, I use stack overflow data from python packages and compare this with the results gotten by the original authors on using this approach on Java APIs.**

***Index Terms*—API documentation, Stack Overflow, Information extraction, insight sentences, Supervised learning**

I. Introduction

Information on how to use Application programming interfaces (APIs) are typically held in a software documentation sources. Documentations are considered the official manual of the APIs. However, despite the availability of API documentations, developers still often encounter problems when using these documentations. This points to the fact that API documentations are either difficult to understand, lacking in information or incomplete in some other way. For example, some API documentation capture information about the functionality and structure of APIs, but offer no information regarding purpose [1]. As such, some of the problems developers face when learning a new API are regarding the APIs documentation [2]

Stack Overflow on the other hand contains a plethora of additional information that are sometimes not in the API documentation. So this project aims to mine such stack Overflow information and devices a method to automatically augment API documentation with insight sentences from stack overflow. The techniques used in this paper were pioneered by Treude et al. [3]. However, in that paper, the researchers used data from java APIs and did not consider whether the research methods developed in that paper will extend to other languages. Thus, in this paper, I examined the result of Augmenting API Documentation with Insights from Stack Overflow using python packages. More specifically, this paper aims to answer the following research questions.

RQ1: To what extent are unsupervised Machine learning approaches able to identify meaningful insight sentences?

RQ2: How does the research method perform when used against APIs from another programming language?

I had initially planned to have a third research question which asked whether "software developers found extracted insight sentences meaningful." However, due to the nature of the term, it was impossible to carry out appropriate surveys to tackle such a research question, so that question was removed from the paper

To answer the first research question, I replicated the supervised insight sentence extractor described in [3]. Here, I used the sentences from stack overflow as well as meta data from stack overflow relating to the sentences as features for the machine learning model.

My main contribution was however in the second research question, where I compared my results with the results from [3] to identify whether this machine learning model yield similar performance when using Python APIs versus when using Java APIs. Unfortunately, due to the small size of the dataset, I was unable to get conclusive results.

II. Motivating example

Fig 1 shows the a few examples of stack overflow posts which have insightful sentences for 3 different python API types. The goal of this paper is to be able to automatically identify such insightful sentences and use these posts containing insightful sentences to augment API documentations.

The first example is an elaborating on what the label encoder in the sklearn.preprocessing.LabelEncoder package does. The other two posts present information that is not explicitly stated in the documentation about how to use the dataframe.apply method and how to change the title of a plot respectively

III. Background

A. Natural Language Processing

Natural Language Processing is the extraction of meaning from human language using a computer [8], [10]. Some of the

API type	Stack overflow answer	Id
sklearn.preprocessing.LabelEncoder	No, labelEncoder does not do this. It takes 1-d arrays of class labels and produces 1-d arrays. It's designed to handle class labels in classification problems, not arbitrary data, and any attempt to force it into other uses will require code to transform the actual problem to the problem it solves (and the solution back to the original space)	24475412
pandas.DataFrame.apply	You can also use df.apply() to iterate over rows and access multiple columns for a function. docs: DataFrame.apply() def valuation_formula(x, y): return x * y * 0.5 df['price'] = df.apply(lambda row: valuation_formula(row['x'], row['y']), axis=1)	30566899
matplotlib.pyplot	To only modify the title's font (and not the font of the axis) I used this: import matplotlib.pyplot as plt fig = plt.Figure() ax = fig.add_subplot(111) ax.set_title('My Title', fontdict={'fontsize': 8, 'fontweight': 'medium'}) The fontdict accepts all kwargs from matplotlib.text.Text.	53745066

Fig. 1. Example of insight sentences

applications of NLP include translation, text retrieval, dialogue systems, information summarization, and categorization [10]. While NLP began as an intersection of AI and Linguistics in the 1950s, it has evolved to use statistical methods and borrows from many diverse fields including Machine Learning, Data Science, and AI [8].

While earlier methods used rules and determinism, newer techniques use probabilities and statistics in the form of machine learning [8]. NLP is a challenging problem due to the complex nature of human language and this means that rule-based systems were limited in what they could achieve.

NLP becomes even more complex when dealing with software artifacts. This is because often times, these artefacts contains lines of codes which can seem like regular natural language, but are very different in terms of syntax and semantics. Depending on the project, some researcher choose to completely ignore code in language processing because code is actually not a natural language. In this paper, I did not ignore code, because there could be some insights like sample code embedded in these code segments.

IV. Related Work

Prior work related to my approach for insight sentence extraction has been largely divided into two categories, namely; improving API documentation and harnessing stack overflow data

A. Harnessing stack overflow data

Preetha Chatter [11] presented a thorough study aimed at understanding how novice software engineers direct their efforts and what kinds of information they focus on within a stack overflow post. In this study, the researchers found that developer read only about 15–21 percent text and about 27 percent code in a post. They put forward this and other results from this study with the hope that these results would be leveraged to provide a better experience for developers on stack overflow

In the paper upon which mine is based, Treude and Robillard [3] created a novel machine learning framework that uses sentences, their formatting, their question, their answer, and their authors as well as part-of-speech tags and the similarity of a sentence to the corresponding API documentation as features to the machine learning model. Seahawk by Bacchelli et al. [4] is another java based approach into tackling API documentation. Seahawk is an Eclipse plug-in that integrates stack overflow content into an integrated development environment (IDE). Seahawk formulates queries from a context in an IDE and presents a list of ranked list of results. Another related tool is prompter, proposed by Ponzanelli et al. [5]. Prompter is designed to automatically retrieve relevant discussions from stack overflow given a context and notify developers about help if necessary. Thus we can see that most of the research in the space used Java APIs and there was need for research on a different language.

B. Improving API documentation

Closely related to SISE is the proposal for integrating crowdsourced FAQs into API documentation by Chen and Zhang [6]. They propose to connect API documentation and informal documentation by capturing information from web browsing history. Other work have has focused on detecting and preventing API errors. Zhong et al. designed DOCREF [7] an approach that combines different methods in the realm of natural language processing to identify inconsistencies in software documentation. The researchers were successful in using DOCREF to identify over a thousand documentation errors.

Several researchers have carried out different projects to improve the usability of API documentations. In [13], Dekel and Herbsleb created an eclipse plugin eMoose that decorates method invocations whose targets have associated directives in API documentation with such directives so that users of the function would be immediately aware of information associated with the use of the function. Similarly, Stylos et al [12] designed a Java API placeholder Jadeite that let users add new “pretend” classes or methods that are displayed in the actual API documentation, and can be annotated with the appropriate APIs to use instead.

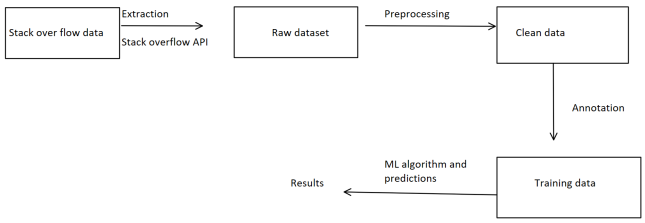

Fig. 2. overview of the process

V. Method

A. Overview

Figure 2 shows an overview of the different steps of this project. I wanted to get a good subset of posts for analysis, but not so many that it would become overly time consuming. Also, because insight sentences are typically contained in answers instead of questions, I only included answers in the dataset. Given these constraints, I chose to limit the data set used in this paper to the 5 most popular python packages namely; pandas, numpy, scipy, matplotlib and scikit-learn. I choose the top 10 answers from top 10 threads relating to these python packages. Since some 10 threads did not have up to 10 answers, the sizes of the dataset for each package were different. To ensure that all the answers gotten were of high quality, I choose only answers with a score greater than 5. The table below shows an overview of the development set.

TABLE I
Python packages

python package	size
numpy	79
sci-kit-learn	62
pandas	147
scipy	75
matplot-lib	86
total	449

B. Data Gathering

1) Collection: I collected data using the Stack Exchange API [9] using carefully constructed SQL queries for each python API. I then merged all the rows of data to form the data set. Unlike the work done by Treude and Robillard [3], I did not split the answers into different sentences as I believe it is easier to find meaning in posts if the entire post is together. As such, I had a small dataset of 449 rows.

2) Queries used: I designed queries to extract the the top 10 answers (with score greater than 5) from top 10 threads relating to the different python packages.

Here is a sample query used to extract data for the numpy package.

```
WITH T1 AS (
    SELECT TOP 10
        *
  FROM Posts AS p2
  WHERE p2.PostTypeId = 1
  AND p2.Tags LIKE '%<python>%'
  AND p2.Tags LIKE '%<numpy>%'


  ORDER BY p2.Score DESC
)

SELECT *
FROM Posts as p1

WHERE exists (

  SELECT *
  FROM T1 as p2
  WHERE p2.id =p1.ParentId
  AND p1.score > 5


)
ORDER By p1.ParentId
```

3) Sampling: To select a sample of the data, first the rows in the data set were randomly shuffled. Then I selected the first 30 rows in the data set and annotated these rows with a label; 1 indicating that the body of the post contained an insightful sentence and 0 indicating that the body of the post did not contain an insightful sentence. I choose such a small sample because of time constraints as annotating the data set was very time intensive.

C. Preprocessing/Cleaning

The posts from stack overflow consists of a lot of extra fluff that needs to be removed before I could proceed, as such, I applied the following steps to clean the data.

1) Converting the body of posts to lower case
2) Remove stop words
3) Stemming using porter stemming
4) Removing columns with unnecessary information Eg. ContentLicense, CommunityOwnedDate etc

D. Feature selection

Due to the time constraint associated with the class, it was infeasible for me to use all the features in the original SISE methodology. As such, I choose to use only a small subset of these feature. I choose features that were easy to extract and feature that had good information gain in the original SISE paper [3]. Table II shows a list of the features I used. The features were, the body of the post,the score of the post, length of the post, Number of comments in the post. Features with greater information gain like cosine similarity between sentence and most similar sentence in API documentation were left out because they were to complex to extract within the scope of this course.

Finally, after the features were selected, I used a bag of words model to convert the Body of the post into numerical data. I used the countVectorizer class of the sklearn.feature_extraction.text for this. When dealing with the body of posts, I intentionally did not remove code elements as code elements could determine whether or not a post is insightful.

TABLE II
DESCRIPTION OF STACK OVERFLOW DATA

Attribute	Description
Id	Unique identifier for post/comment
Body	Content of the post
Tags	Denotes the topics the question relates to
score	The score assigned to the post
comment count	Number of comments the post hast

E. Machine learning model

After the data has been cleaned and features have been selected, I applied the random forest algorithm to the training data using 60 percent of the annotated data for training.

VI. RESULTS

Unfortunately, the results gotten from the machine learning model was disappointing. Figure 3 shows a summary of said result. The accuracy, precision and f1 score for class 0 (posts with no insight sentences) is 0.750, 1.0, 0.857. While for class 1, the precision, accuracy and F1 score is 0, 0, 0

	precision	recall	f1-score
0	0.750	1.000	0.857
1	0.000	0.000	0.000

Fig. 3. Results

VII. DISCUSSION

Since the results of the machine learning model were so unsatisfactory, I decided to investigate why this was the case. On inspection, I discovered that the annotated dataset was unbalanced and contained only a few posts with insightful sentences. Coupled with the small size of the annotated dataset, the classifier was unable to get any meaningful results for insightful sentences. Because of the poor performance of the classifier, it is impossible to tangibly answer the posed research questions.

VIII. THREATS TO VALIDITY

The quality of this research project ended up being very far from that of the original project because of time constraints associated with the semester. As such, there were some simplifications that were taken to reduce the complexity of the project that might affect the validity of this project.

The biggest threat to validity was the small dataset used. Since machine learning algorithms perform better with larger datasets, with an annotated dataset of only 30 rows and only 18 rows for training, the validity of the results with such a small dataset could be questioned. Also, when annotating the dataset, I independently choose whether or not a post contains an insightful sentence. This label associated with a post is subjective and a different researcher might annotated some posts in the dataset differently. So annotation of the dataset presents another threat to validity. Under different circumstances, a better approach would be to leave the annotation to different independent software developers and only choose data points where they agree with each other.

Finally, another threat to validity is the fact that the features used for the machine learning model were so few that the most important features (cosine similarity with API documentation) were omitted. However, this was unavoidable as it would have been too time intensive to calculate the cosine similarities for all posts.

IX. FUTURE WORK

As state of the art techniques in natural language processing continues to change, ongoing research needs to continue using these state of the art techniques to support usability of API documentation. Also, more extensive work needs to go on investigating the performance of state of the art techniques against a more robust suite of languages instead of just Java and python. As such, this paper can be seen as a first step into an inquiry into the effectiveness of SISE techniques when used with other languages

Also, the features and machine learning techniques used in this paper can be extend to augument API documentation using data from other sources like github or reddit. Since these websites are also brimming with plenty of software developer information, we can apply similar techniques to mine insightful sentences from these sources. Going even further, we can extend this work to mine insight sentences from other textual artifacts produced by software developers like bug reports, commit messages or code comments.

X. GITHUB

The link to the github hosting page can be found here https://github.com/ooghre/Augument_API_Documentation

REFERENCES

[1] W. Maalej and M. P. Robillard. Patterns of knowledge in API reference documentation. *IEEE Trans. on Software Engineering,* 39(9):1264-1282, 2013.

[2] M. P. Robillard and R. Deline. A field study of API learning obstacles. *Empirical Software Engineering,* 16(6):703-732, 2011.

[3] C. Treude and M. Robillard. Augmenting API Documentation with Insights from Stack Overflow. 2016 10.1145/2884781.2884800.

[4] A. Bacchelli, L. Ponzanelli, and M. Lanza. Harnessing Stack Overflow for the IDE. In Proc. of the 3rd Int'l. *Workshop on Recommendation Systems for Software Engineering,* pages 26-30, 2012.

[5] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza. Mining Stack Overflow to turn the IDE into a self-confident programming prompter. *In Proc. of the 11th Working Conf. on Mining Software Repositories,* pages 102-111, 2014.

[6] C. Chen and K. Zhang. Who asked what: Integrating crowdsourced FAQs into API documentation. *in Companion Proc. of the 36th Int'l. Conf. on Software Engineering,* pages 456-459, 2014.

[7] H. Zhong and Z. Su. Detecting API documentation errors. *In Proc. of the Int'l. Conf. on Object Oriented Programming Systems Languages and Applications,* pages 803-816, 2013.

[8] P. M. Nadkarni, L. Ohno-Machado, and W. W. Chapman, "Natural language processing: an introduction," *Journal of the American Medical Informatics Association,* vol. 18, no. 5, pp. 544–551, 2011.

[9] "Browse Queries," Stack Exchange Data Explorer. [Online]. Available: https://data.stackexchange.com/stackoverflow. [Accessed: 07-Dec-2020].
[10] C. A. Thompson, "A Brief Introduction to Natural Language Processing for Non-linguists," *Learning Language in Logic Lecture Notes in Computer Science,* pp. 36–48, 2000.
[11] K. Chatterjee. "Finding Help with Programming Errors: An Exploratory Study of Novice Software Engineers' Focus in Stack Overflow Posts." *The Journal of Systems and Software,* vol. 159, Elsevier BV, Jan. 2020, p. 110454–, doi:10.1016/j.jss.2019.110454.
[12] F. Stylos. "Improving API Documentation Using API Usage Information." *2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), IEEE,* 2009, pp. 119–26, doi:10.1109/VLHCC.2009.5295283.
[13] D. Herbsleb. "Improving API Documentation Usability with Knowledge Pushing." *Proceedings of the 31st International Conference on Software Engineering, IEEE Computer Society,* 2009, pp. 320–30, doi:10.1109/ICSE.2009.5070532.