ASHESI UNIVERSITY

THESIS

CONTENT VS. METRICS: USING LANGUAGE MODELING TO EVALUATE IN-LINE SOURCE CODE COMMENTS FOR PYTHON

UNDERGRADUATE THESIS

B.S.c Computer Science

Maame Efua Boham
2020

ASHESI UNIVERSITY

CONTENT VS. METRICS: USING LANGUAGE MODELING TO EVALUATE IN-LINE SOURCE CODE COMMENTS FOR PYTHON

UNDERGRADUATE THESIS

Undergraduate Thesis submitted to the Department of Computer Science, Ashesi University in partial fulfilment of the requirements for the award of Bachelor of Science degree in Computer Science

Maame Efua Boham

2020

DECLARATION

I hereby declare that this Undergraduate Thesis is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature: ....

Candidates Name: ....

Date: ....

I hereby declare that preparation and presentation of this Undergraduate Thesis were supervised in accordance with the guidelines on supervision of thesis laid down by Ashesi University.

Supervisor's Signature: ....

Supervisor's Name: ....

Date: ....

Acknowledgement

This thesis was completed only with the help, support and contributions of many people. It is unfortunately impossible to name them all. However, I would like to recognize, with utmost gratitude, some specific individuals for their contributions to the success of this thesis.

I would like to express my deepest gratitude to my supervisor, Mr. Dennis Asamoah-Owusu, for his endless patience and invaluable contribution to the realization of this project. I would also like to say a huge thank you to my friends and family for their unending support.

I would, finally, like to dedicate this paper to my best friend, Ariel Woode, without whom this thesis would never have seen the light of day.

Abstract

Documentation is vital to the understanding, maintenance and, ultimately, survival of software projects . And yet, a lot of software projects either lack documentation, or are very poorly documented. This results in a gradual decline in the quality of the code and may require complete overhauls in extreme cases. It is therefore important to evaluate documentation to ensure that it conveys clear and meaningful ideas. While existing methods of evaluating documentation are metrics based and look at the structure of documentation examples, this paper explores the possibility of evaluating documentation by assessing its contents. There is, however, a lack of an existing corpus of documentation for natural language processing tasks. A corpus of Python function/method comments is assembled, and a language modeling experiment is performed on them. The results of this experiment are mixed. While they show that it is possible to evaluate documentation by looking at its content as opposed to structure, they also show that this approach may not necessarily be more accurate, with lower quality comment examples having higher probability than those of higher quality.

Keywords: *documentation evaluation; nlp; language modeling; function descriptions*

Contents

1 Introduction and Background

For any developer, technical documentation is an integral part of adjusting to a new system. The more comprehensive the documentation, the less time it takes a new developer to learn how to use a software or piece of code. The quality of documentation also affects the maintainability of the source code or system[19]. Unfortunately, developers have a tendency to neglect documentation, or write it in a way that is not comprehensive or detailed enough to truly be user-friendly[23]. In addition, when new code is added to existing source code, there is a high likelihood that the comments, which serve as source code documentation, will not be updated to reflect the update[15].

When documentation is poor, it results in the degradation of the quality of the code in question[19]. It also reduces the likelihood of the code or system to be stable through upgrades[3].

In order to tackle the issue of the lack of documentation, several approaches have been utilized. These include automated documentation generation[22], the encouragement of self-explanatory code and selecting priority areas for which developers must absolutely write consistent and complete documentation[23].

Even with advances in tools used in the documentation process and automation, it is still necessary for documentation to be evaluated in order to truly assess whether it is capable of meeting its goal of imparting knowledge to potential users[16]. In organizations, for example, the onus would then fall on team managers to review documentation [if any] that was being submitted along with code by a developer. This documentation can be reviewed or evaluated in several ways depending on the structure and size of the organization, as well as the programming languages being used.

Currently, this documentation is measured as opposed to being evaluated. Open source tools such as 'Docstr-Coverage' [24] for Python allow developers to measure documentation coverage by detecting which modules in source code lack documentation. This however does not stem the occurrences of redundant documentation in which developers state the obvious. Docstring ratios while a step in the right direction, are not able to effectively capture how comprehensive or user-friendly a piece of documentation is for given source code.

Correspondingly, in the field of Natural Language Processing there have been several developments in the creation of tools that can automatically analyze and perform other operations on software artifacts, i.e. interpret natural language text such as documentation and code comments[31]. Using Natural Language Processing tools to evaluate documentation may lead to more informative and accurate results when testing. There is however, as at this time, no existing corpus of documentation for use in NLP tasks.

This paper will be focused on Python source code comments for function declarations as the specific type of documentation to be experimented on. In addition, good quality and comprehensive will be used interchangeably as descriptors of documentation, while poor quality and non-comprehensive will be used interchangeably as well.

1.1 Research Questions

Based on these observations, the objective of this paper is to propose a system that uses Natural Language Processing (NLP) tools to analyze and evaluate documentation to test for comprehensiveness. The hypothesis being tested is as follows:

1. Is it possible to compile a reliable corpus of documentation for use in Natural Language Processing?

2. Will a system using Natural Language Processing be able to judge the comprehensiveness and user-friendliness of a piece of documentation by using the content and not metrics?

3. Will it result in a more accurate depiction?

2 Related Work

2.1 Importance of documentation

From the health and technology to the justice sectors, documentation is a necessary tool[33]. This documentation must, however, be diligently curated in order to be useful to potential users. When it comes to source code, studies have shown that not only is documentation the second-most-used artefact for code understanding[29], it vastly simplifies maintenance as well[17]. Documentation that is poorly written results in misunderstandings and difficulty in maintenance[18].

2.2 How documentation is being evaluated currently

Another way of evaluating documentation or technical text at the moment is a metric-based measurement system. This still requires manual evaluation by assessors which is time-consuming and results in increased cost. Also, metric systems for evaluating text, while easy to implement, may result in oversights when it comes to the actual meaning of the text. For example, a metric for completeness, as detailed by Aversano, Guardabascio and Tortorella[4], would be calculated as a ratio of the number of classes in the code to the number of classes described in the documentation. While this gives an idea on whether every class has been documented, it does not account for ambiguity within the documentation or the general level of understandability, user-friendliness or comprehensiveness shown.

Automatic quality assessment of comments has also been explored, with the JavaDoc Miner notably being developed for Java[21]. While this project evaluates quality of language of the source code comments and the consistency between source code and its corresponding comments, it does so by using various heuristics and measurements such as readability heuristics using the Flesch, Fog and Kincaid Indexes. It also takes into account measurements such as the words per Javadoc comment and the abbreviation count per comment[21]. Although these are valid and certainly useful estimations of the quality of source code comments, they are attempts at measuring the quality of the comment structure as opposed to evaluating the contents of the comments themselves.

2.3 Benefits of Natural Language Processing in Software Engineering

Software engineers/developers create natural language documents at many points in the software development life cycle. These documents may include anything from user manuals to code commit messages and source code documentation. In order to extract useful information from these documents, Natural Language Processing tools geared towards technical text have been developed. An example of such a tool is QuARS, which is used to analyze natural language requirements documents[14]. The automation of this task has improved the development life cycle.

2.4 Choosing an NLP Library/Technique

In selecting NLP libraries for use on technical documents, it is important to keep in mind the complexities that accompany such documents and choose relevant libraries in accordance with those complexities[2]. Research shows that the use of varying libraries can affect the results of the analysis being done[2]. One technique used in the analysis of technical documentation is the Functional Analysis technique. This technique expresses interactions as Subject-Action-Object triples and has been used notably for functional analysis of patents as well as the analysis of software requirement documents[10]. For the purposes of this research, Keras and its accompanying libraries will be used as they are simple to implement[34].

2.5 Defining documentation quality

Documentation is an important tool in creating value for users[25,30]. For documentation to maximize value, it needs to be comprehensive and of good quality. In the past, defining the metrics for documentation quality has required the assessment of various ideas on quality such as documentation length and ease of search[5]. Various metrics-based systems have also been implemented, studying metrics such as completeness, readability and structure of documentation[4].

1. Completeness: this describes the ability of the documentation to describe all items of the source code, i.e. classes, methods and packages[4].
2. Readability: this examines the degree to which documentation expresses clear and understandable concepts[4] and is calculated using the Flesch Readability Index.
3. Structure: this evaluates the structure of the documentation as regards the number of chapters, sections and sub-sections, document length, density of tables and figures et cetera[4].

In evaluating documentation from a Natural language perspective, a few ideas from the evaluation of software requirements written in natural language can be applied. These are found in Meyer's Seven Sins of The Specifier[13] which look at conceptual understanding of requirement text.

2.6 Documentation that makes sense/ language modeling

A language model is a probabilistic mechanism for generating text[12]. It can also be defined as a conditional distribution on the identity of the i-th word in a sequence, given the identity of previous words[8]. Historically, language models have been used in several natural language processing tasks such as Machine Translation, Speech Recognition, Spelling Correction, Optical Character Recognition, Text Summarization and many others[9]. In recent years, language modeling has evolved to include neural network based models and these Neural Language Models (NLMs) achieve much better results than classical language models[9].

The use of language modeling in documentation is also not novel as language models have been used in predicting programming comments[26]. Statistical language models have also been used to predict reading difficulty of generic texts sourced from the web[11]. The goal of this paper is to follow on the same path by applying these concepts in order to evaluate whether documentation, specifically, in-line comments, make logical sense.

Building on the idea of readability as detailed by Aversano, Guardabascio and Tortorella[4], this paper seeks to evaluate the degree to which in-line source code documentation expresses clear and logical concepts by calculating for the probability of the sentence using language modeling.

3 Approach and Methodology

The methodology for this paper can be outlined in four simple steps:

1. Defining a quality indicator

2. Building a corpus of good quality examples to train the language model

3. Testing the model on documentation examples classified as good quality and bad quality

4. Evaluating the results

3.1 Defining a Quality Indicator

In this paper, in order to evaluate documentation, the content of the documentation is assessed. To achieve this, it is necessary to explicitly describe what good quality documentation looks like. Generally, in the evaluation of software engineering related documents, there are several indicators used for assessing documentation quality including ambiguity, conciseness, vagueness and others. Regarding this paper, quality is determined as:

1. The documentation follows a logical pattern and expresses a clear and logical concept. For example:
   'Given a file name, read the file, retrieve the stories, and then convert the sentences into a single story.' This example of documentation conveys a clear idea, is semantically correct and is logical to a reader.

Great strides have been made in using NLP, specifically language modeling, to detect anomalies or the level of correctness and probability of sentences. If a sentence or a documentation example makes clear and logical sense, it will have a high probability when passed through a language model[32]. It also means that for each example of function describing documentation, in order for it to be assessed as quality documentation, every word in the example must have a high likelihood of following its predecessor. This can be represented using the Markov Assumption as:

$$P(w_1 w_2 ... w_n) \approx \prod_i P(w_i | w_{i-k} ... w_{i-1})$$

3.2 Sourcing of Training and Testing Documentation

3.2.1 Conditions for sourcing documentation

1. To ensure that the sourced documentation is accurately labeled, a manual sanity check will be performed on random samples from the sourced data.

2. At least 2000 sentence examples will be collected for each type of documentation to be evaluated.

3. Only one type of documentation must be used for training and testing to avoid inaccurate comparison.

In order to accurately classify documentation as either comprehensive or not comprehensive, this paper sources two types of documentation for testing.

3.2.2 Documentation that can be classified as comprehensive

This is defined as documentation that makes logical sense. For the purposes of simplicity, this documentation is sourced from six notable open source projects with a large following and a rigorous pipeline for accepting contributions:

1. Keras [34]

2. NumPy [35]

3. SciPy [36]

4. Django [37]

5. Flask [38]

6. Scikit-learn [39]

It is assumed that as a result of the process documentation and code contributions must go through before they are accepted, they are of good quality. This paper assembled a corpus of good quality in-line function documentation containing 15,001 examples. To source this documentation, a Python program was written that cloned the given GitHub projects, traversed the cloned projects, and for every file in the folder, retrieved the method descriptions. The data was then stored in a .DATA file to be used in training and testing the model. Libraries used here were the Python subprocess, os, ast and pickle libraries. The pseudo-code for sourcing the documentation is below:

```
$NODETYPES \leftarrow ast.FunctionDef :' Function/Method$
cloneProjects
$urls \leftarrow$ list of github urls
$index \leftarrow 1$
$root \leftarrow$ os.getcwd()

while urls:
newChild $\rightarrow$ name of project
create new directory and change into it
run git clone on url using subprocess
os.chdir(root) change directory to root
index += 1

function recursiveFileFinder(root):
initialize functionDescriptions as List
j $\rightarrow$ 0
for each item in os.walk(root):
for every file in item[2]:
rest,extension $\rightarrow$ os.path.splitext(file)
if ext is “.py”:
functionDescriptions.extend(pyParser(os.path.join(item[0], file)))
return functionDescriptions

function pyParser(filePath):
open filePath as inputFile:
source $\rightarrow$ inputFile.read()
initialize class descriptions as List

tree $\rightarrow$ ast.parse(source)
for every node in ast.walk(tree):
if isinstance(node, tuple(NODETYPES):
docstring $\rightarrow$ ast.get$_d ocstring(node)$
$ifdocstring :$
$appenddocstringtofunctionDescriptions$

return functionDescriptions

function datasetCreation(functionDescriptions):
open new .DATA file as outputFile
pickle.dump(functionDescriptions)
```

Table 1: Examples of Comprehensive Documentation

Number	Examples
1	’Parse stories provided in the bAbi tasks format If ’only supporting’ is true, only the sentences that support the answer are kept.’
2	‘Extract the module docstring.finds the line at which the docstring ends.’
3	’Copy the examples directory in the documentation.files by extracting the docstrings written in Markdown.’
4	’One-hot encode given string C. ArgumentsC: string, to be encoded.numrows: Number of rows in the returned one-hot encoding. This isused to keep the of rows for each data the same.’
5	’Given a file name, read the file,the stories,then convert the sentences into a single story.If maxlength is supplied,stories longer than maxlength tokens will be discarded.’
6	”Return the tokens of a sentence including punctuation.»> tokenize(’Bob dropped the apple. Where is the apple?’)[’Bob’, ’dropped’, ’the’, ’apple’, ’.’, ’Where’, ’is’, ’the’, ’apple’, ’?’]”

3.2.3 Documentation that can be classified as not comprehensive

This is defined as documentation that does not make logical sense. That includes incomplete sentences. This documentation will be sourced from personal projects on GitHub that are under the GPL or LGPL licenses and fit the criterion of not making logical sense. The documentation that was classified as not comprehensive was sourced by building a scraper for GitHub URLs using Python’s Beautiful Soup and Request libraries[28,40].

Table 2: Examples of Non-Comprehensive Documentation

Number	Examples
1	””” document’s write method ”””
2	” doc.meta.addElement(dc.Publisher(text=publisher) doc.meta.addElement(dc.Rights(text=copyrights))”
3	”In the middle of a paragraph”
4	””” constructor ”””
5	”” set for download asociated file ””
6	””” MoinMoin used || to delimit table cells”””

3.3 Experiment

3.3.1 Objective

To determine that documentation makes sense, this paper will make use of language modeling to calculate for the probability of a given sentence or sequence of words in a class description.

3.3.2 Corpus

Table 3: Corpus

	Comprehensive Examples	**Non-Comprehensive Examples**	**Total**
Training	7500	-	7500
Testing	7501	33	7534
Total	15001	33	15034

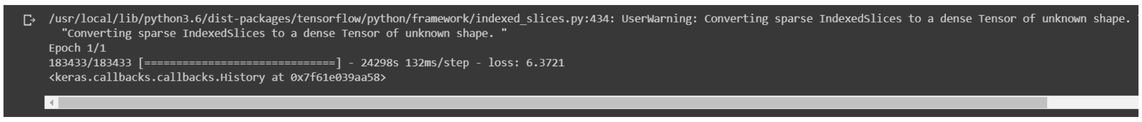
3.3.3 Data Processing

In order for the model to train on the test data that has been gathered, the data must first be processed into a format that can be understood by the model. This involves tokenization to break down the examples into smaller units and one-hot encoding to map the tokens to vectors[20,41].

3.3.4 Model

The language model was built using a Keras' Sequential model and a Long Short-Term Memory (LSTM) Recurrent Neural Network (RNN). A Neural Language Model (NLM) was used because of its ability to produce good results on sequences of varying lengths[27]. LSTMs are excellent for use in language models because of their ability to capture long term memory[42].

The model also consists of an embedding layer and a dense layer with a SoftMax activation function, RMSprop optimizer and a categorical cross entropy loss function[43]. The model was then trained on the corpus.



```
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slices.py:434: UserWarning: Converting sparse IndexedSlices to a dense Tensor of unknown shape.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
Epoch 1/1
183433/183433 [==============================] - 24298s 132ms/step - loss: 6.3721
<keras.callbacks.callbacks.History at 0x7f61e039aa58>
```

Figure 3.6 - Completed Training of Model

The model performed fairly well with a final loss value of 6.3721.

3.3.5 Calculating Probability

The code to calculate the probability of the sentences was obtained from an open source project on GitHub[1] and optimized to suit the purposes of this paper. The function returns the probability of each example that is passed to it. To test the hypothesis of this paper, a list of test data that is considered comprehensive and a list of test data that is considered not comprehensive were passed to the function, and the results stored. The minimum and maximum probabilities of both test data sets , as well as the average probability and variance are also calculated.

4 Results

To avoid disparity, and considering the limited non-comprehensive data, the test sample for probability calculations was limited to 30 examples for both comprehensive and non-comprehensive in-line source code comments. However, the available examples for testing were 2000 examples of comprehensive examples and 33 examples of non-comprehensive examples as shown in Figure 4 below.

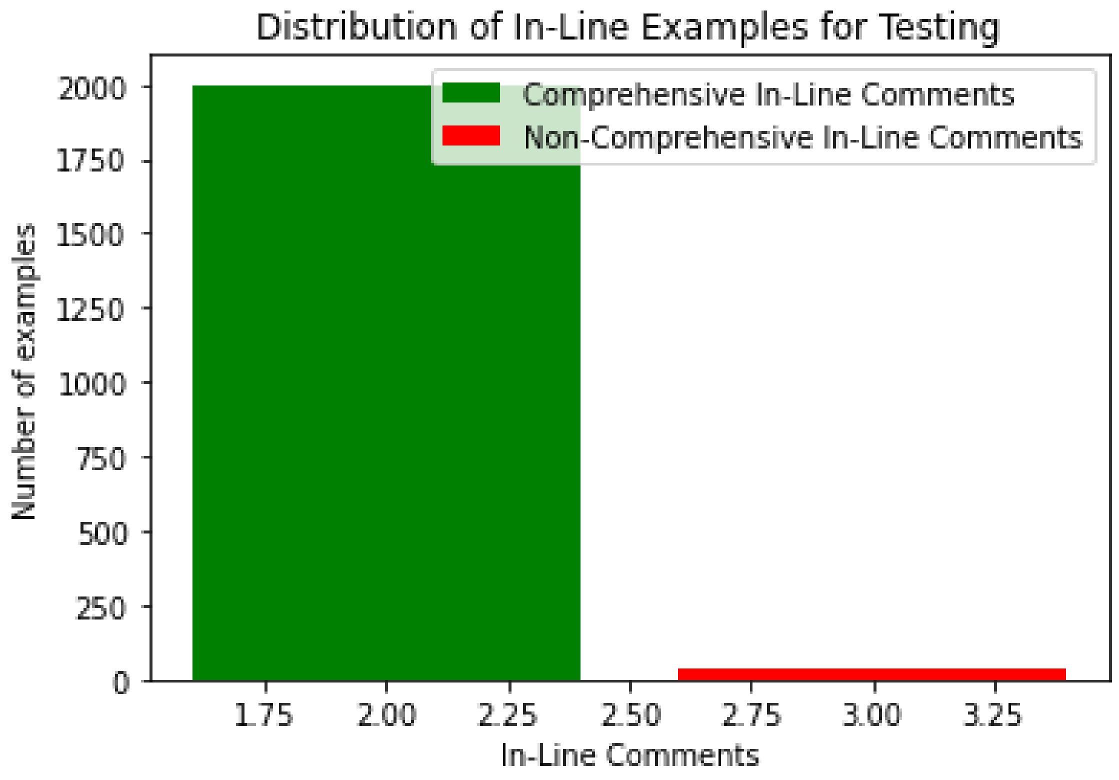


Figure 4

The statistics calculated on the test data were:

- Maximum probability of good and bad test data
- Minimum probability of good and bad test data
- Average probability of good and bad test data
- Variance in probability of good and bad test data

Non-comprehensive comments had a higher minimum probability of 1.3048789e-72 than comprehensive data's 0.0 probability. Non-comprehensive comments also had a higher maximum probability than comprehensive data, with 8.079122833e-05 as compared to 3.60583077e-09 for comprehensive data.

This means that the model believes non-comprehensive data is makes more sense or is more probable to occur than comprehensive data.

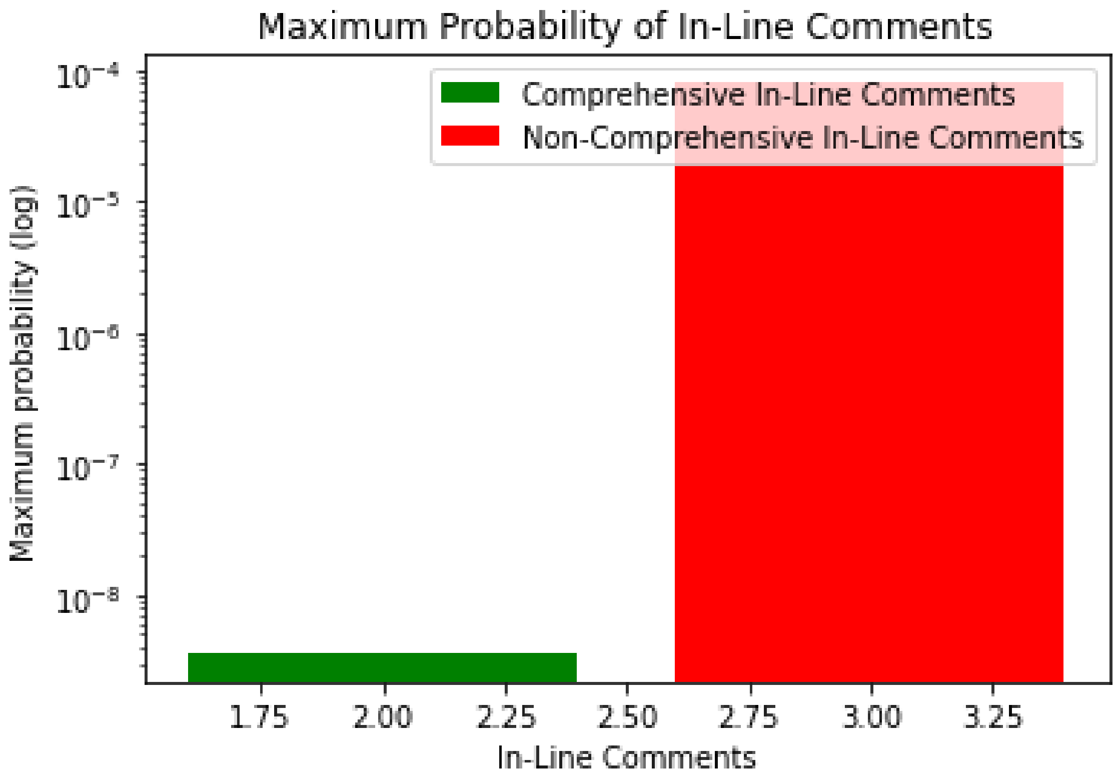


Figure 5

There was much larger variance in the non-comprehensive test data set than the comprehensive data set as well, with the comprehensive data set registering a lower average than the non-comprehensive test data.

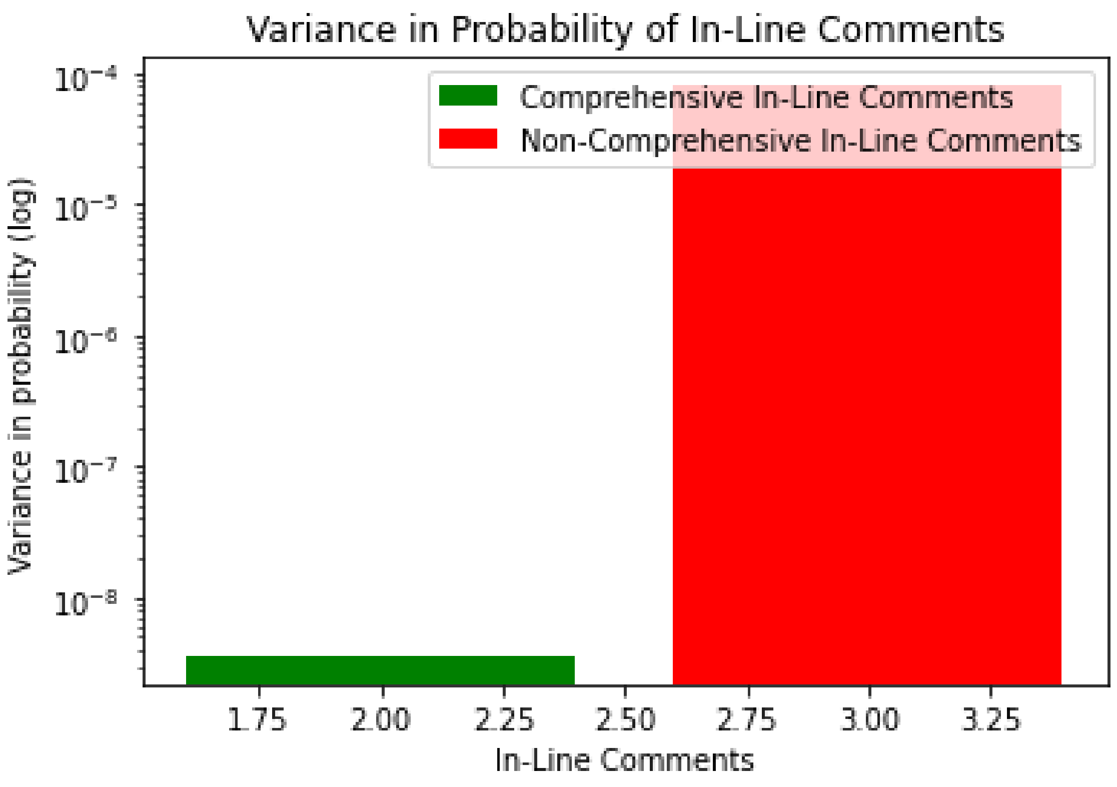


Figure 5

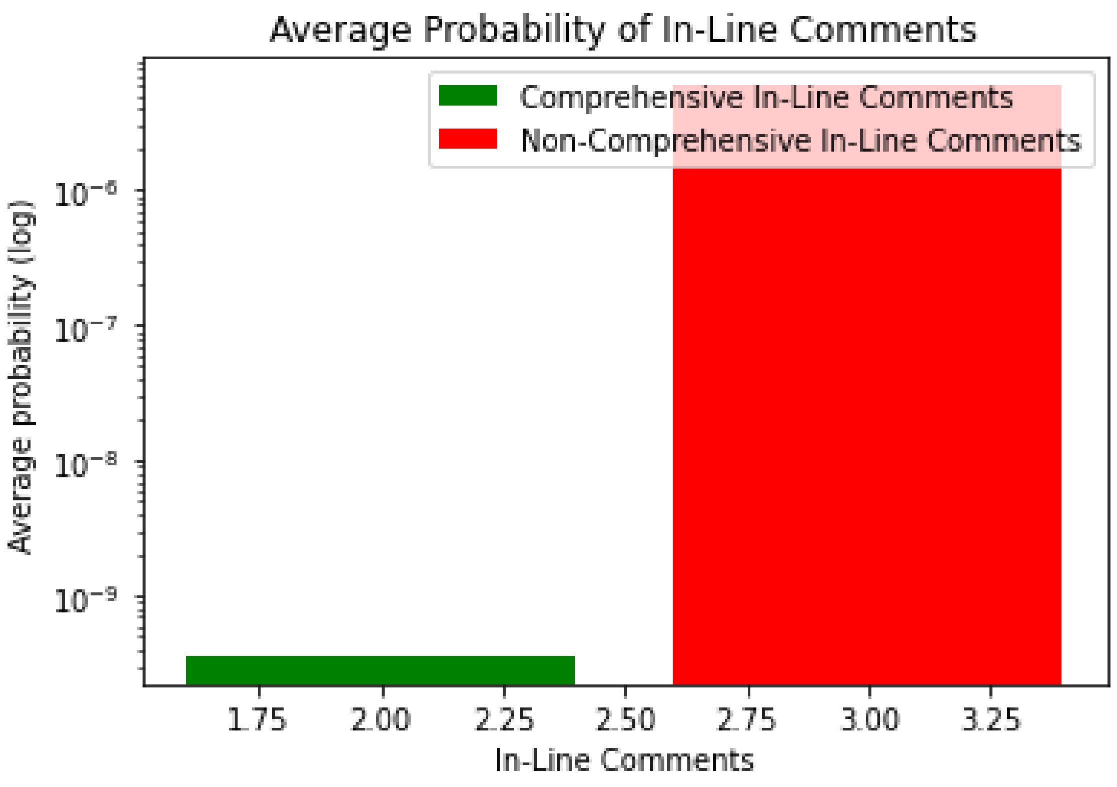


Figure 5

These are certainly fascinating results. It would be expected that comprehensive in-line comments for source code would generally have higher probability than non-comprehensive in-line comments, considering that the idea of language modeling evaluates the actual content of the examples, as stated in the research question and explored through the literature review. These results, however, do not confirm the second hypothesis, which seeks to garner more accurate results using Natural Language Processing. It must, however, be noted that the examples in the corpus of comprehensive in-line comments have longer sequences than those of the non-comprehensive in-line comments. This affects the performance of the LSTM-based model as LSTMs can be difficult to use when there are long input sequences with a single output[7]. This outweighs the fact that the model was trained on examples of comprehensive examples. More accurate results may be gained through the use of other Natural Language Processing tools.

5 Conclusion and Recommendations

5.1 Summary

This paper sought to build a corpus of documentation examples for use in Natural Language Processing, to explore the idea of evaluating documentation using its contents as opposed to various metrics, and to determine whether that method of evaluation would provide accurate results. A corpus of comprehensive documentation was built by extracting source code comments from notable open source projects. A much smaller collection of non-comprehensive examples was assembled for the purposes of this paper from open source personal projects on GitHub. A language model was built using an LSTM and trained on a portion of the comprehensive data and then tested on both comprehensive and non-comprehensive data by calculating for the probability of each example.

The results of the experiments were mixed with the non-comprehensive documentation having a higher maximum and minimum probability, average probability and variance than comprehensive examples. While it does prove that documentation can be evaluated on its content and semantics as opposed to structure and metrics, it also shows that this evaluation may not necessarily be more accurate. The research in this paper, is however, a solid steppingstone for future research into the evaluation of documentation based on the inherent meaning it conveys to users. It may be possible to use other natural language processing tools, even in the field of language modeling, that take into account the length as well as the meaning of the given examples.

5.2 Limitations and Design Flaws

A major limitation of this paper was the lack of a GPU, which resulted in the use of Google Colab to train and test the language model. While Colab proved immensely useful, it is designed to time out after 90 minutes of idleness and this affected the training of the model which took several hours.

A lack of data was another limitation of this paper. More examples to train the model would reduce the risk of over-fitting to the available data and more testing examples would provide more insights from the results.
A design flaw of the experiment was the fact that it does not take into account how length of examples affects their probability, especially when working with an LSTM model.

5.3 Suggestions for Future Work

5.3.1 Building a Larger Corpus of Not-Comprehensive Documentation

In its attempt to build a corpus of documentation to be used for NLP tasks, this paper was able to source 15,000 examples of comprehensive documentation. For non-comprehensive information, however, the numbers are much lower. A higher number of examples could lead to more informative results.

5.3.2 Further Cleaning/Pre-processing

In order to achieve better results in the experiment, further cleaning can be done of the assembled corpus. The 15,034-word corpus of documentation does not take into account commented out code, which would be recognized as comments. This may affect the average probability of the comments in the corpus. In addition, the corpus does not remove the ending of line (EOL) marker '/n'. While this may not affect the model as it was trained on documentation containing these markers, it would produce cleaner data for training and testing. Generally, more pre-processing of the data is necessary before use in the model.

5.3.3 Using SciBERT

While there does not currently exist a standard corpus of quality documentation, re-running this experiment using SciBERT, a version of Google's BERT model that has been pretrained on scientific text, may prove beneficial[6]. It includes cased and uncased versions and provides state-of-the-art performance on several NLP tasks in the scientific domain. This model may then be fine-tuned using the assembled corpus of documentation to improve performance and results.

BIBLIOGRAPHY

[1] 262588213843476. Computing the probability of occurrence of a sentence with a LSTM model using Keras. *Gist*. Retrieved May 10, 2020 from https://gist.github.com/rvinas/cf5c4c47456834d7fd4e3328858cffe2

[2] Fouad Nasser A Al Omran and Christoph Treude. 2017. Choosing an NLP Library for Analyzing Software Documentation: A Systematic Literature Review and a Series of Experiments. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, IEEE, Buenos Aires, Argentina, 187–197. DOI:https://doi.org/10.1109/MSR.2017.42

[3] Hirohisa Aman and Hirokazu Okazaki. 2008. Impact of Comment Statement on Code Stability in Open Source Development. In *Proceedings of the 2008 conference on Knowledge-Based Software Engineering: Proceedings of the Eighth Joint Conference on Knowledge-Based Software Engineering*, IOS Press, NLD, 415–419.

[4] Lerina Aversano, Daniela Guardabascio, and Maria Tortorella. 2017. Evaluating the Quality of the Documentation of Open Source Software: In *Proceedings of the 12th International Conference on Evaluation of Novel Approaches to Software Engineering*, SCITEPRESS - Science and Technology Publications, Porto, Portugal, 308–313. DOI:https://doi.org/10.5220/0006369403080313

[5] Hanna Bandes. 1986. Defining and Controlling Documentation Quality: Part I. *Tech. Commun.* 33, 1 (1986), 6–9.

[6] Iz Beltagy, Kyle Lo, and Arman Cohan. 2019. SciBERT: A Pretrained Language Model for Scientific Text. *ArXiv190310676 Cs* (September 2019). Retrieved May 1, 2020 from http://arxiv.org/abs/1903.10676

[7] Jason Brownlee. 2017. Techniques to Handle Very Long Sequences with LSTMs. *Machine Learning Mastery*. Retrieved May 11, 2020 from https://machinelearningmastery.com/handle-long-sequences-long-short-term-memory-recurrent-neural-networks/

[8] Jason Brownlee. 2017. Gentle Introduction to Statistical Language Modeling and Neural Language Models. *Machine Learning Mastery*. Retrieved May 4, 2020 from https://machinelearningmastery.com/statistical-language-modeling-and-neural-language-models/

[9] Jason Brownlee. 2017. Gentle Introduction to Statistical Language Modeling and Neural Language Models. *Machine Learning Mastery*. Retrieved February 9, 2020 from https://machinelearningmastery.com/statistical-language-modeling-and-neural-language-models/

[10] Gaetano Cascini, Alessandro Fantechi, and Emilio Spinicci. 2004. Natural Language Processing of Patents and Technical Documentation. In *Document Analysis Systems VI* (Lecture Notes in Computer Science), Springer Berlin Heidelberg, 508–520.

[11] Kevyn Collins-Thompson and Jamie Callan. 2005. Predicting reading difficulty with statistical language models. *J. Am. Soc. Inf. Sci. Technol.* 56, 13 (November 2005), 1448–1462. DOI:https://doi.org/10.1002/asi.20243

[12] Bruce Croft and John Lafferty. 2003. *Language Modeling for Information Retrieval*. Springer Science & Business Media.

[13] Elj. 2014. eljeiffel: The seven sins of a specifier - Bertrand Meyer / IEEE SOFTWARE (Jan 1985). *eljeiffel*. Retrieved December 18, 2019 from http://eljeiffel.blogspot.com/2014/04/the-seven-sins-of-specifier-bertrand.html

[14] F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami. 2001. The linguistic approach to the natural language requirements quality: benefit of the use of an automatic tool. In

Proceedings 26th Annual NASA Goddard Software Engineering Workshop, 97–105. DOI:https://doi.org/10.1109/SEW.2001.992662

[15] Beat Fluri, Michael Wursch, and Harald C. Gall. 2007. Do Code and Comments Co-Evolve? On the Relation between Source Code and Comment Changes. In *Proceedings of the 14th Working Conference on Reverse Engineering* (WCRE '07), IEEE Computer Society, USA, 70–79. DOI:https://doi.org/10.1109/WCRE.2007.21

[16] Andrew Forward and Timothy C. Lethbridge. 2002. The relevance of software documentation, tools and technologies: a survey. In *Proceedings of the 2002 ACM symposium on Document engineering* (DocEng '02), Association for Computing Machinery, McLean, Virginia, USA, 26–33. DOI:https://doi.org/10.1145/585058.585065

[17] Dorsaf Haouari, Houari Sahraoui, and Philippe Langlais. 2011. How Good is Your Comment? A Study of Comments in Java Programs. In *2011 International Symposium on Empirical Software Engineering and Measurement*, 137–146. DOI:https://doi.org/10.1109/ESEM.2011.22

[18] Carl S. Hartzman and Charles F. Austin. 1993. Maintenance productivity: observations based on an experience in a large system environment. In *Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: software engineering - Volume 1* (CASCON '93), IBM Press, Toronto, Ontario, Canada, 138–170.

[19] Mira Kajko-Mattsson. 2005. A Survey of Documentation Practice within Corrective Maintenance. Retrieved April 20, 2020 from https://doi.org/10.1023/B:LIDA.0000048322.42751.ca

[20] Vimarsh Karbhari. 2020. Why do we need one-hot encoding? *Medium*. Retrieved May 11, 2020 from https://medium.com/acing-ai/why-do-we-need-one-hot-encoding-7bcb456d49df

[21] Ninus Khamis, René Witte, and Juergen Rilling. *Automatic Quality Assessment of Source Code Comments: The JavadocMiner*.

[22] Paul W. McBurney. 2015. Automatic Documentation Generation via Source Code Summarization. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, 903–906. DOI:https://doi.org/10.1109/ICSE.2015.288

[23] Paul W. McBurney, Siyuan Jiang, Marouane Kessentini, Nicholas A. Kraft, Ameer Armaly, Mohamed Wiem Mkaouer, and Collin McMillan. 2018. Towards Prioritizing Documentation Effort. *IEEE Trans. Softw. Eng.* 44, 9 (September 2018), 897–913. DOI:https://doi.org/10.1109/TSE.2017.2716950

[24] Hunter McGushion. *docstr-coverage: Utility for examining python source files to ensure proper documentation. Lists missing docstrings, and calculates overall docstring coverage percentage rating*. Retrieved October 14, 2019 from https://github.com/HunterMcGushion/docstr_coverage

[25] JAY MEAD. 1998. Measuring the Value Added by Technical Documentation: A Review of Research and Practice. *Tech. Commun.* 45, 3 (1998), 353–379.

[26] Dana Movshovitz-Attias and William W Cohen. Natural Language Models for Predicting Programming Comments. 6.

[27] Guozheng Rao, Weihang Huang, Zhiyong Feng, and Qiong Cong. 2018. LSTM with sentence representations for Document-level Sentiment Classification. *Neurocomputing* (May 2018). DOI:https://doi.org/10.1016/j.neucom.2018.04.045

[28] Kenneth Reitz. *requests: Python HTTP for Humans.* Retrieved May 11, 2020 from https://requests.readthedocs.io

[29] Sergio Cozzetti B. de Souza, Nicolas Anquetil, and Káthia M. de Oliveira. 2006. Which documentation for software maintenance? *J. Braz. Comput. Soc.* 12, 3 (October 2006), 31–44. DOI:https://doi.org/10.1007/BF03194494

[30] Gholamreza Torkzadeh. 1988. The Quality of User Documentation: An Instrument Validation. *J. Manag. Inf. Syst.* 5, 2 (1988), 99–108.

[31] Christoph Treude, Carlos A. Prolo, and Fernando Figueira Filho. 2015. Challenges in Analyzing Software Documentation in Portuguese. In *Proceedings of the 2015 29th Brazilian Symposium on Software Engineering* (SBES '15), IEEE Computer Society, Washington, DC, USA, 179–184. DOI:https://doi.org/10.1109/SBES.2015.27

[32] ChengXiang Zhai. 2008. *Statistical Language Models for Information Retrieval*. Now Publishers Inc.

[33] THE IMPORTANCE OF ACCURATE DOCUMENTATION - ProQuest. Retrieved May 1, 2020 from https://search.proquest.com/openview/8b8bf55cf173efdf5c604eb2a66334d7/1?pq-origsite=gscholar&cbl=33490

[34] Keras: the Python deep learning API. Retrieved May 11, 2020 from https://keras.io/

[35] NumPy — NumPy. Retrieved May 11, 2020 from https://numpy.org/

[36] SciPy.org — SciPy.org. Retrieved May 11, 2020 from https://www.scipy.org/

[37] The Web framework for perfectionists with deadlines | Django. Retrieved May 11, 2020 from https://www.djangoproject.com/

[38] Welcome to Flask — Flask Documentation (1.1.x). Retrieved May 11, 2020 from https://flask.palletsprojects.com/en/1.1.x/

[39] scikit-learn: machine learning in Python — scikit-learn 0.22.2 documentation. Retrieved May 11, 2020 from https://scikit-learn.org/stable/

[40] Beautiful Soup Documentation — Beautiful Soup 4.9.0 documentation. Retrieved May 11, 2020 from https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[41] Tokenization. Retrieved May 11, 2020 from https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html

[42] Understanding LSTM Networks -- colah's blog. Retrieved April 8, 2020 from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[43] machine learning - How to build a Language model using LSTM that assigns probability of occurence for a given sentence. *Stack Overflow*. Retrieved May 10, 2020 from https://stackoverflow.com/questions/51123481/how-to-build-a-language-model-using-lstm-that-assigns-probability-of-occurence-f