Automatically improving natural language in software documentation using Stack Overflow edits

Project Report

Max Iraci-Sareri
School of Computer Science
The University of Adelaide
`a1771834@student.adelaide.edu.au`

Supervisor: Christoph Treude
School of Computer Science
The University of Adelaide
`christoph.treude@adelaide.edu.au`

Supervisor: Sebastian Baltes
School of Computer Science
The University of Adelaide
`sebastian.baltes@adelaide.edu.au`

Supervisor: Markus Wagner
School of Computer Science
The University of Adelaide
`markus.wagner@adelaide.edu.au`

2020/11/05

Abstract

The scope of software documentation resources includes both official resources like API documentation and crowd-sourced websites such as Stack Overflow. Threads on Stack Overflow contribute greatly to documentation but are of varying quality. Posts on Stack Overflow are publicly editable, with users optionally providing descriptive edit messages. These edits are categorised in the paper 'An Annotated Dataset of Stack Overflow Post Edits'[2].

To improve them, this project develops and evaluates a prototype model to improve basic post readability using the results from this dataset. This was achieved primarily through use of natural language processing and machine learning, alongside trivial readability improvements. The resulting program can fix trivial issues such as sentence casing and formatting, and also warns users when often-redundant information is added to posts or documentation.

1 Introduction

Software documentation assists a wide range of programmers from every skill level; it ensures the software is easy to understand and program for. Due to its nature as an instructional tool, it must be perceived as authoritative to ensure reliability and trust.

Good software documentation can take time to write, so often gets crowd-sourced through popular community-led sites such as Stack Overflow. While this improves the quantity of documentation, the quality suffers; it can be poor in terms of readability. In order to keep information clear, concise, and reliable, existing Stack Overflow posts are refined by community members through edits. These post edits are all documented, and a portion of them have descriptive 'edit messages', often used to note what was changed in the edit.

The paper 'An Annotated Dataset of Stack Overflow Post Edits'[2] takes this data and categorises it by what was mentioned as a change in the edit message. These categorised edits are explored in this project to apply their use in the context of improving existing Stack Overflow posts in terms of usability and readability.

2 Project Motivation

While software exists to improve the quality of existing code, there is not much in terms of software used to improve readability and usability in existing software documentation. The goal of this project is to provide a usable prototype as a proof of concept to demonstrate that Stack Overflow posts (and, by extension, software documentation) can be improved through natural language processing and by use of machine learning to categorise the data.

3 Literature Review

3.1 Natural Language Processing

Natural language processing (NLP) techniques are important in processing of user-generated content; Stack Overflow edit messages and the edits they convey do not have a mandated style to follow. NLP has been investigated towards usage in software documentation before, including with Stack Overflow [8]. The study (Treude et al., 2017) found that most traditional NLP libraries were not efficient at parsing software documentation correctly due to improper treatment of tokens and syntax. From investigation they found the best NLP library to use was the SpaCy python library; from a manual random subset of 1,116 tokens generated from analysing Stack Overflow text it performed with an accuracy of 90%. Furthermore, another study conducted to use NLP techniques and machine learning libraries [1](Alreshedy et al, 2018) used Stack Overflow posts and code samples.

3.2 Stack Overflow Edits

Considerable progress has been made in the pursuit of using Stack Overflow edits. Many of these pursuits use the SOTorrent database; a set of Stack Overflow posts built to analyse the evolution of posts through their edits. Diamantopoulos et al. investigated multiple series of edits to Stack Overflow posts [5] to find common patterns in the edits; they suggested that the research could be used to recommend edits in future posts, but did not implement or create such a tool. Similarly, Tang et al. [11] used Stack Overflow edits to investigate whether edits and edit-messages together could be used as training data to fix potential bugs in external code; in their experiment, much like this project, they categorised types of edits to attempt to find the most useful for fixing bugs. Furthermore, Yin et. al [15] proposed a method to train a neural network to 'capture the correlation between NL and code', with the possibility of scaling their system beyond the programming languages they used to test.

3.3 Software Documentation

Improvement of software documentation is also a heavily-tread topic; other approaches often similarly use Stack Overflow as a source of training data. Robillard et al. [12] used a machine-learning approach to create the Supervised Insight Sentence Extractor (SISE), which analyses posts from Stack Overflow and finds 'insight sentences' which can be used to augment software documentation. This project takes a similar approach by attempting to provide useful augmentations of software documentation directly.

Uddin et al. [13] found the three severest problems with software documentation were ambiguity, incompleteness, and incorrectness of content.

3.4 Related Work

Stack Overflow is used commonly as a source of research and tool development. Yang et. al [14] specifically used code snippets in StackOverflow posts to determine how usable each snippet on the site is, with the goal of 'developing automated tools with the Stack Overflow snippets and surrounding text'.

Nasehi et. al [7] used categorisation of Stack Overflow posts to determine that 'the explanations accompanying examples are as important as the examples themselves', further proving the necessity of well-written documentation accompanying syntax examples.

Aside from Stack Overflow, there have been attempts to use NLP to analyse other publicly editable online sources of information such as Wikipedia [4], where Daxenberger et al. classify edit categories in Wikipedia revisions. These edit categories are used to detect vandalism and improve site quality through labelling problematic pages.

This project has a similar framework to these works; building on top of existing Stack Overflow categorisation [2](Baltes et al., 2020) allows us to immediately identify fixes that could be models for future machine learning training.

4 Methodology

4.1 Tools and Software Used

For testing and training machine learning models alongside natural language processes, rapid iteration and testing is required. For this reason, Python 3.8 was chosen alongside the Python library Jupyter Notebook[6], which assisted in both presentation and iteration of the data. Furthermore, many Python libraries for machine learning and natural language processing exist; this allows for compatibility between segments of the program, saving time and effort. The main libraries used included `pandas`[10] for data analysis, `scikit-learn`[9] for machine learning, and `nltk`[3] for natural language processing, and `textblob` for spelling correction.

4.2 Data Used

The data used was sourced from the dataset used in the paper 'An Annotated Dataset of Stack Overflow Post Edits'. The sample used to train the data consisted of 10487 unique post revisions, with corresponding `ContentBefore` and `ContentAfter` fields, alongside which category of edit was found in the edit message.

4.3 Difference Tool

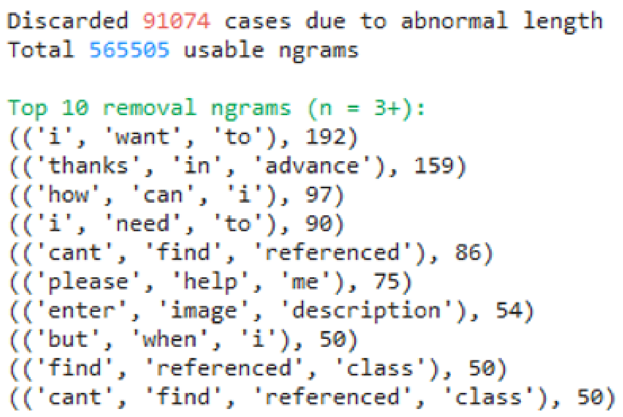
Python by default does not have a tool to cleanly display the difference between two strings. This project uses the `difflib` library to compute differences between the strings, and the `colorama` library to display any differences in colour. The differences generated through `difflib` categorise each difference as additions, removals, or updates to the string, which are all represented in this project's 'diff tool' through different colours. It also generates a ratio of similarity between the two strings, which is used in this project as part of automated testing.

4.4 Natural Language Processing

4.4.1 Redundant Phrase Removal

The primary goal for natural language processing in this project was to warn the user of any phrases that commonly get removed as part of edits to Stack Overflow posts. If prevented before the need for an edit, it may improve the quality of the text at a glance.

The method for identifying commonly-removed phrases was to first find the n-grams that appeared the most in edits marked as 'removing content'. This was done by iterating over the dataset, using the `difflib` library to compare the `ContentBefore` and `ContentAfter` fields. For any text that was computed as ‘removed’ or ‘updated’, the ‘before’ field was tokenised and n-grams of the text were calculated for an n value of 3 to 7. Any removed text that was shorter than three characters long could not form a meaningful n-gram and was discarded.



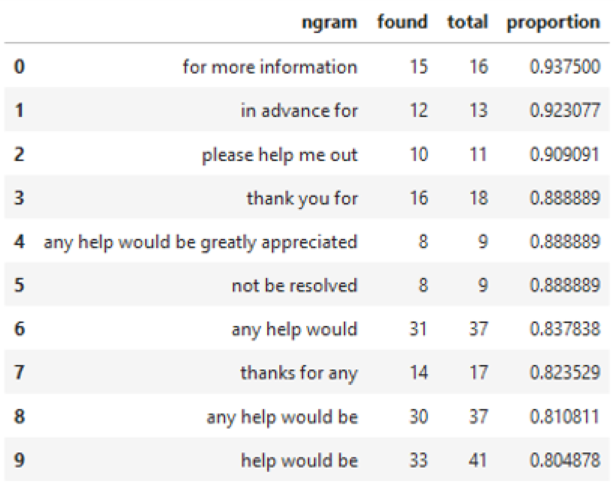
```
Discarded 91074 cases due to abnormal length
Total 565505 usable ngrams

Top 10 removal ngrams (n = 3+):
(('i', 'want', 'to'), 192)
(('thanks', 'in', 'advance'), 159)
(('how', 'can', 'i'), 97)
(('i', 'need', 'to'), 90)
(('cant', 'find', 'referenced'), 86)
(('please', 'help', 'me'), 75)
(('enter', 'image', 'description'), 54)
(('but', 'when', 'i'), 50)
(('find', 'referenced', 'class'), 50)
(('cant', 'find', 'referenced', 'class'), 50)
```

Figure 1: The 10 most common n-grams found based on the removal data ($n \geq 3$)

While this data shows conclusively what n-grams are removed the most from the entire dataset, it does not show how proportionally they are removed; for example, the n-gram ‘i want to’ (shown in Figure 1) is considered the ‘most removed’, but realistically would not be consistently removed regardless of context.

To obtain the ‘most commonly removed’ n-grams, the data was searched once again, aiming to find all the n-grams in the dataset regardless of whether the diff-tool marked them as ”removals”. The totals found were then compared to how often each was marked as a removal, creating a proportion for each n-gram that was used to sort the data.

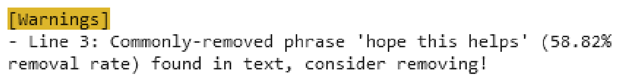


	ngram	found	total	proportion
0	for more information	15	16	0.937500
1	in advance for	12	13	0.923077
2	please help me out	10	11	0.909091
3	thank you for	16	18	0.888889
4	any help would be greatly appreciated	8	9	0.888889
5	not be resolved	8	9	0.888889
6	any help would	31	37	0.837838
7	thanks for any	14	17	0.823529
8	any help would be	30	37	0.810811
9	help would be	33	41	0.804878

Figure 2: The 10 most common n-grams based on how proportionally they were removed during edits to Stack Overflow posts ($n \geq 3$)

This sorted data (Figure 2) displays a distinct difference to the earlier result. Now the phrases that are present in the top 10 are phrases that realistically provide no meaningful input towards documentation. With this data, the n-gram ‘i want to’, while previously showing up as ‘most removed’, now shows as removed in only 14.7% of edits.

With the proportional n-grams identified, they were saved to a .csv file for future use, preventing the need for recalculation. During operation, the prototype reads the input text line-by-line, and attempts to find any of the n-grams inside it. The system prioritises higher values of n where possible; for example, the phrase ‘any help would be’ would be prioritised over ‘help would be’ as higher values of n usually mean higher specificity. If any of the n-grams are found in the input that have more than a 50% removal rate, a warning is generated encouraging the user to remove the phrase (Figure 3).



```
[Warnings]
- Line 3: Commonly-removed phrase 'hope this helps' (58.82%
removal rate) found in text, consider removing!
```

Figure 3: A warning generated through the prototype, suggesting the removal of the phrase ‘hope this helps’.

4.4.2 Spelling Correction

Natural language processing was also used in the project for correcting spelling. This is done word-by-word, instead of in the context of the sentence; as such it is more limited than a more thorough approach. It cannot fix grammatical issues like tense or improper homophones as a result, as it lacks the context of the surrounding words.

The text is scanned using regular Python, comparing each word against both a list of English words and a list of Stack Overflow tags. The list of tags is necessary to encapsulate lots of technical jargon that gets added to posts; without it, words like ‘YouTube’ get corrected to ’couture‘. Once a potentially-misspelled word is found, the `textblob` library is used to find potential replacements. If `textblob` finds a suitable replacement, it is replaced, and a message is logged mentioning the change. If no suitable replacement is found, the program naively assumes that the word it has found is part of inline code, and encapsulates it in Markdown inline code tags. This was the simplest way in the project to identify inline code in StackOverflow posts; however, due to this simplicity it often finds false positives.

4.5 Machine Learning - Code Identification

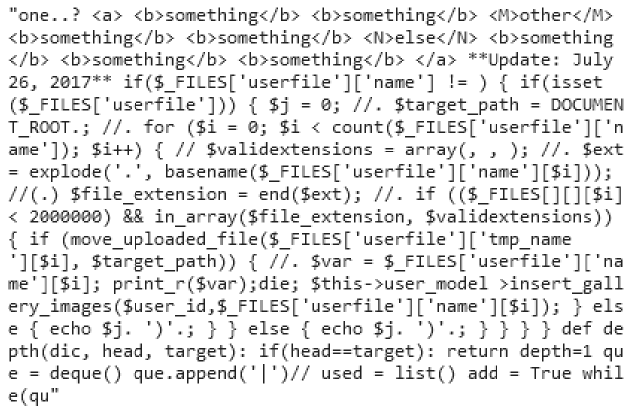
4.5.1 Training the Model

The goal of machine learning in this project is to find and appropriately tag code snippets in Stack Overflow with Markdown tags. Often posters on Stack Overflow will copy and paste their error message or code without formatting it correctly, which makes it hard to read and answer.

The machine learning model trained for this project took samples of code taken from the same categorised dataset used for everything else. To gather training data in order to categorise text as ‘code’ or ‘not code’, the dataset was filtered appropriately. Text was considered ‘code’ if it had fewer than 10 spaces per line, alongside whether it contained a semicolon at the end of the line. Any pre-existing code-snippets (marked with the s̀ymbol) were also included as code. A sample of this can be seen in Figure 4.

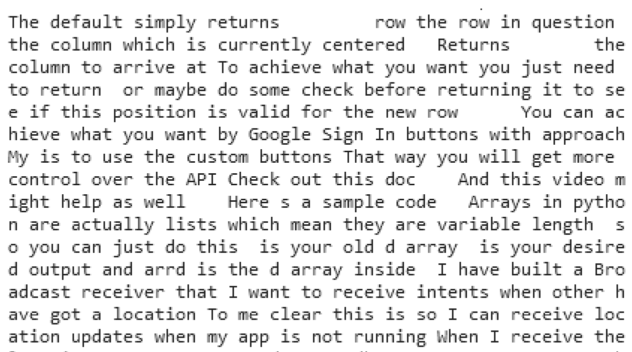
Text in the dataset was considered ‘not code’ when it contained more than 10 spaces per line and/or included singular capital letters (such as the capital letter ‘I’). URLs were also searched for and removed, as they appear frequently in Stack Overflow posts. To get pure text, most punctuation and digits were removed; this has the limitation of inaccurately removing quotation marks from words. A sample of text considered ‘not code’ can be seen in Figure 5.

Both sets of training data removed any newline characters, tab characters, and any non-ASCII Unicode symbols for a string of words usable in the model.



```
"one..? <a> <b>something</b> <b>something</b> <M>other</M>
<b>something</b> <b>something</b> <N>else</N> <b>something
</b> <b>something</b> <b>something</b> </a> **Update: July
26, 2017** if($_FILES['userfile']['name'] != ) { if(isset
($_FILES['userfile'])) { $j = 0; //. $target_path = DOCUMEN
T_ROOT.; //. for ($i = 0; $i < count($_FILES['userfile']['n
ame']); $i++) { // $validextensions = array(, , ); //. $ext
= explode('.', basename($_FILES['userfile']['name'][$i]));
//(.) $file_extension = end($ext); //. if (($_FILES[][][$i]
< 2000000) && in_array($file_extension, $validextensions))
{ if (move_uploaded_file($_FILES['userfile']['tmp_name
'][$i], $target_path)) { //. $var = $_FILES['userfile']['na
me'][$i]; print_r($var);die; $this->user_model >insert_gall
ery_images($user_id,$_FILES['userfile']['name'][$i]); } els
e { echo $j. ')'.; } } else { echo $j. ')'.; } } } } def de
pth(dic, head, target): if(head==target): return depth=1 qu
e = deque() que.append('|')// used = list() add = True whil
e(qu"
```

Figure 4: A short sample of the type of text considered in the training data as ‘code’.



```
The default simply returns         row the row in question
the column which is currently centered   Returns        the
column to arrive at To achieve what you want you just need
to return  or maybe do some check before returning it to se
e if this position is valid for the new row      You can ac
hieve what you want by Google Sign In buttons with approach
My is to use the custom buttons That way you will get more
control over the API Check out this doc    And this video m
ight help as well    Here s a sample code   Arrays in pytho
n are actually lists which mean they are variable length  s
o you can just do this  is your old d array  is your desire
d output and arrd is the d array inside  I have built a Bro
adcast receiver that I want to receive intents when other h
ave got a location To me clear this is so I can receive loc
ation updates when my app is not running When I receive the
location I start an to update my db  However I am unsure wh
ich intent filter to use in the manifest  Is there a list o
f what intent filters I can use I have a TimeZone as NSStri
```

Figure 5: A short sample of the type of text considered in the training data as ‘not code’.

4.5.2 Code Identification

The model was trained using the ‘Native Bayes classifier’ model in the `scikit-learn` library. The Native Bayes classifier is a probabilistic categoriser that attempts to predict categories of text depending on provided training sets of similar data.

The tagging system reads through the code and feeds each line into the model, which gives its prediction as to whether the line is code or not. The model is set to automatically ignore any lines starting with a preceding angle bracket, as those are most commonly used in Markdown and Stack Overflow as indication of a stack trace or console output. It also skips any existing code snippet which already have the Markdown formatting around them, as false-positive code identifications are rare on Stack Overflow. Finally, it skips any lines with square-bracketed digits, as these are most likely image tags. After the processing, if a line is identified as ‘containing code’, it is wrapped in Markdown tags.

4.6 Trivial Changes

Trivial changes involve improvements that do not require machine learning or natural language processing to function. Simple Python functions and text replacements can fix most of the problems classified as ‘Formatting’ in the dataset. Examples of these include sentence casing and Markdown replacements; ordered lists in Stack Overflow require a period rather than a round bracket, and that is easily replaceable through regular expressions and text replacement functions.

5 Experimental Setup

5.1 Manual Testing

Manual testing was performed on a case-by-case basis based on personal opinion. The tests were made against a dataset of 150 randomly-selected individual posts. These posts were divided into 30 of the following categories: ‘Formatting’, ‘Grammar’, ‘Adding’, ‘Editing’, and ‘Improving’. Tests was considered a success as considered against the question: ’Did the edits made improve the post?’ As this is inherently opinionated, the tests were performed with some leniency; errors in the result that were present before the prototype ran were considered less important than errors fixed by the prototype.

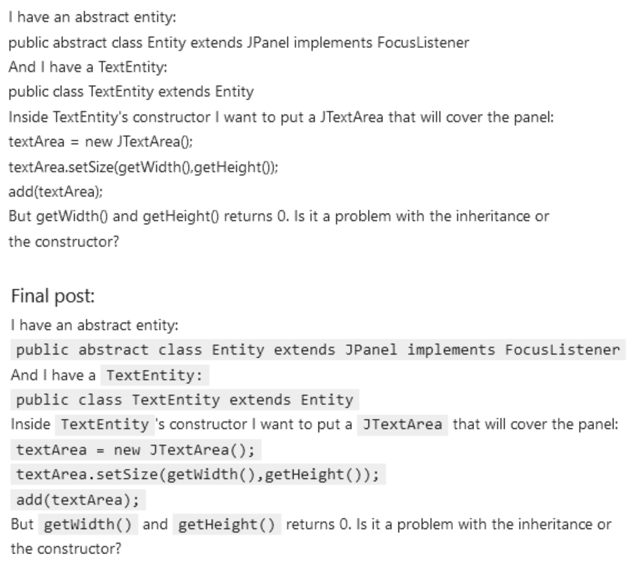
5.2 Automated Testing

Similar to the manual method, automated testing was performed based on 150 random posts of the same categories. The similarity was automated by comparing what was changed by the prototype to the actual edit on Stack Overflow. The similarities were measured using the `difflib` library with the aforementioned ‘similarity ratio’; if this ratio value was over 0.85 (signifying the two strings were over 85% similar), it was considered a success.

6 Results

6.1 Comparisons

Figures 6 and 7 include hand-picked examples of successful documentation improvement.


I have an abstract entity:
public abstract class Entity extends JPanel implements FocusListener
And I have a TextEntity:
public class TextEntity extends Entity
Inside TextEntity's constructor I want to put a JTextArea that will cover the panel:
textArea = new JTextArea();
textArea.setSize(getWidth(),getHeight());
add(textArea);
But getWidth() and getHeight() returns 0. Is it a problem with the inheritance or the constructor?

Final post:
I have an abstract entity:
`public abstract class Entity extends JPanel implements FocusListener`
And I have a `TextEntity:`
`public class TextEntity extends Entity`
Inside `TextEntity` 's constructor I want to put a `JTextArea` that will cover the panel:
`textArea = new JTextArea();`
`textArea.setSize(getWidth(),getHeight());`
`add(textArea);`
But `getWidth()` and `getHeight()` returns 0. Is it a problem with the inheritance or the constructor?

Figure 6: A comparison between the sample input text and the formatted output text. This shows automatic tagging of code snippets as well as inline code formatting.

6.2 Test Results

6.2.1 Manual Testing

As the manual testing (Figure 8) was based on opinion rather than exact cases, the results performed better than the automated testing. Most of the posts in the 'Formatting' category performed well because the problems with the initial post were fixed by the prototype; most of the issues in posts tagged 'Formatting' had poor sentence casing or missing code tags. Some edge cases prevented all of the posts from performing well; this is expanded further in Section 6.3.

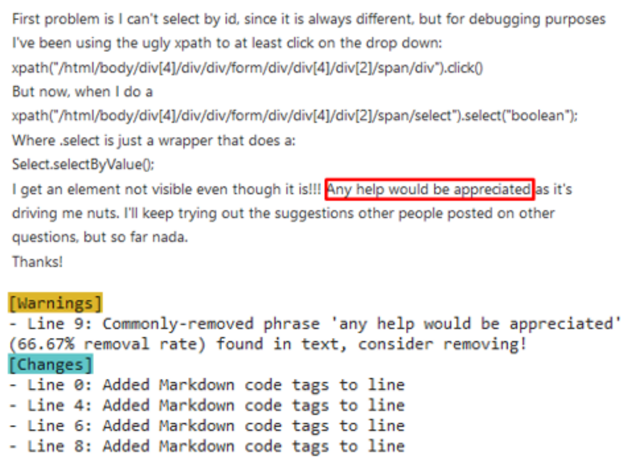
The 'Grammar' category performed well when fixing spelling errors, but also for different reasons; often the original editor did not fix the sort of issues that the prototype does. In this case the prototype performs successfully where a human may have not. However, since the prototype does not perform grammar replacements, the prototype would not have done enough in each case to be considered 'properly edited'.

Edits in the 'Adding' category had varying levels of success. The prototype only works on the content before the edit; therefore, any text added afterwards had no formatting applied to it. This often meant that the prototype would try to fix issues that were not present, sometimes 'fixing' correct formatting.

This problem was amplified in the 'Editing' and 'Improving' categories. The significant issues with the existing documentation did not get fixed by the prototype; formatting and replacement suggestions did not fix the more problematic underlying issues in need of edits.

6.2.2 Automated Testing

The harsher rules of the automated testing (Figure 9) meant that many of the posts were not considered 'successful'. Most cases performed worse, as the data did not change where necessary. Similar to manual testing, 'Formatting' performed well because the prototype performed the same changes that the actual edit did: Markdown improvements and/or sentence casing. 'Grammar' performed surprisingly well in the automated testing given that its success in the manual tests relied on subjective improvement. This may be due to favourable test cases or minimal changes in the actual edit. The 'Improving' category performed similarly badly in the automated testing; the edit would change grammar rules that either were not originally incorrect or largely unimportant compared to the more pressing issues. The 'Adding' category performed the worst in automated testing as the added edit content would be significantly different to whatever the prototype would output.


First problem is I can't select by id, since it is always different, but for debugging purposes I've been using the ugly xpath to at least click on the drop down:
xpath("/html/body/div[4]/div/div/form/div/div[4]/div[2]/span/div").click()
But now, when I do a
xpath("/html/body/div[4]/div/div/form/div/div[4]/div[2]/span/select").select("boolean");
Where .select is just a wrapper that does a:
Select.selectByValue();
I get an element not visible even though it is!!! Any help would be appreciated as it's driving me nuts. I'll keep trying out the suggestions other people posted on other questions, but so far nada.
Thanks!

```
[Warnings]
- Line 9: Commonly-removed phrase 'any help would be appreciated'
(66.67% removal rate) found in text, consider removing!
[Changes]
- Line 0: Added Markdown code tags to line
- Line 4: Added Markdown code tags to line
- Line 6: Added Markdown code tags to line
- Line 8: Added Markdown code tags to line
```

Figure 7: A sample of the warnings and change log generated from the sample input text. This example shows the natural language processing at work; the highlighted commonly-removed phrase 'any help would be appreciated' is found and a warning is sent encouraging the user to remove it.

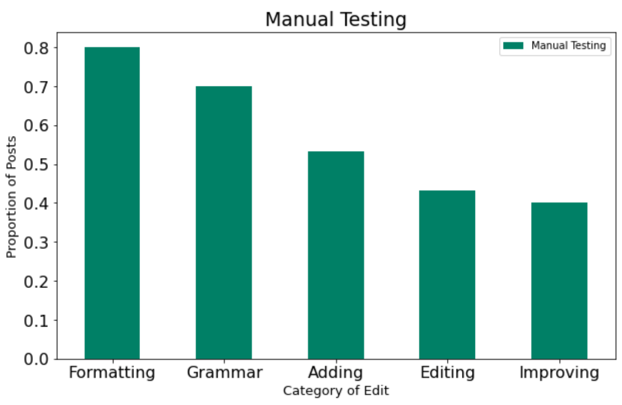


Figure 8: The results of manual testing of the processed text, sorted by category ($n = 30$)

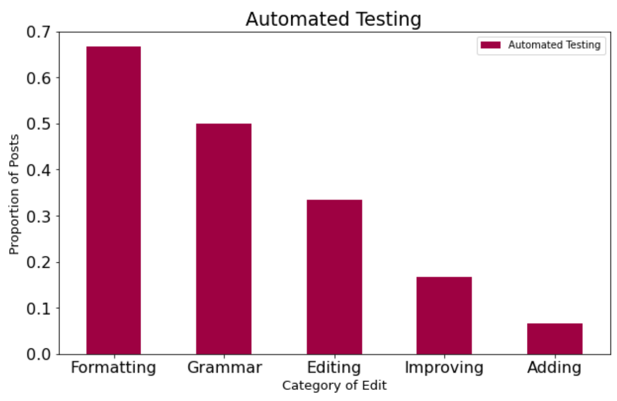


Figure 9: The results of automated testing of the processed text, sorted by category ($n = 30$)

6.3 Limitations/Issues

6.3.1 False Cases

The prototype contains a lot of corner cases that prevent full use in a general context. The trivial corrections (especially for Markdown corrections) often trigger too often, turning valid brackets into periods. This could be fixed with more care with the patterns used in the regular expressions. Similarly when spell-checking, the checker will correct any word it finds a valid replacement for, even if the confidence of the replaced value is very low. This also has a very trivial fix that remained unexplored in the scope of this project.

6.3.2 Prescriptivism

Many of the false cases that occur are due to a design decision rather than a single piece of bad code. Early on in the project, a decision was made to attempt to rewrite and correct any section of text identifiable as being in need of change wherever possible. This, while a noble effort, did not lead to good results. More often than not, words can be corrected into context-incorrect replacements; this can lead to situations where the word pre-correction can be more readable than the replacement that was changed. Furthermore, some words that are commonly identifiable by a human reader did not show up in the list of English words used. The word "interactable" is an example of one such word that a human reader would have no issue understanding but that the computer did not properly identify.

In future, an attempt should be made to fully identify poor grammar and spelling in the context of a sentence; more importantly, however, the code should be descriptive rather than prescriptive until it has a considerable success rate.

6.3.3 Scope

The largest limitation in the project was one of scope; while it corrects certain cases to do with formatting and warns about other cases to do with removed phrases, it makes no attempt to fix sentence-scale errors, or even improperly capitalised words. Also, since it interprets any word that is not found in the word list as an inline code snippet, it can often pick up improper formatting as a spelling error rather than a formatting error.

The original scope of the project did include fixes for grammar and many of the issues outlined above, but was changed due to time constraints and difficulty; see Section 8.1.

7 Repository Access

The GitHub repository for the project can be found at `https://github.cs.adelaide.edu.au/a1771834/stack_overflow_post_improver`

8 Conclusion

The prototype system works well enough as a proof of concept; however, it would need to be heavily refined for use in correcting actual posts. It does not detect or correct enough of the more important readability issues in sentences such as poor sentence structure. With more effort, the project could eventually be able to reconstruct posts to be more clear and contain more readable natural language.

8.1 Project Changes

The original project's plan was to 'improve readability of documentation by removing ambiguity and incompleteness where possible'; this scope is not only far larger than the time frame that supposedly represents it but also is rather unattainable due to its vagueness. Actually restructuring the posts seen on Stack Overflow would require the entire sentence to be de-constructed using natural language processing and re-constructed to be correct, which has not been explored enough in similar work to be feasible as only a portion of this project. Furthermore, to fix incompleteness would be to ask a program to understand the context of a post; at best, this means incorporating an interpreter for every possible code context; at worst, this means knowing exactly what the original poster actually meant by each post.

8.2 Future Work

If given more time, the project could be expanded through more natural language processing replacements. For example, sentences containing problematic n-grams could be automatically fixed instead of simply warned against.

Furthermore, a browser plugin or extension could be programmed, potentially with the help of the Tampermonkey or Greasemonkey browser extensions; these extensions could hook into the JavaScript segments of Stack Overflow itself, allowing the potential warnings to be displayed in realtime over posts. It could be used as a tool for moderators or editors of Stack Overflow posts to signify what portions of existing posts should be edited.

References

[1] Kamel Alreshedy, Dhanush Dharmaretnam, Daniel M. German, Venkatesh Srinivasan, and T. Aaron Gulliver. Predicting the programming language of questions and snippets of stackoverflow using natural language processing, 2018.

[2] Sebastian Baltes and Markus Wagner. An annotated dataset of Stack Overflow post edits, 2020.

[3] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python.* O'Reilly Media, 2009.

[4] Johannes Daxenberger and Iryna Gurevych. *Automatically Classifying Edit Categories in Wikipedia Revisions.* 2013.

[5] Themistoklis Diamantopoulos, Maria Ioanna Sifaki, and Andreas Symeonidis. Towards mining answer edits to extract evolution patterns in Stack Overflow. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR).* IEEE, May 2019.

[6] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, Carol Willing, and Jupyter development team. Jupyter notebooks - a publishing format for reproducible computational workflows. In Fernando Loizides and Birgit Scmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87–90, Netherlands, 2016. IOS Press.

[7] Seyed Mehdi Nasehi, Jonathan Sillito, Frank Maurer, and Chris Burns. What makes a good code example?: A study of programming Q&A in StackOverflow. In *2012 28th IEEE International Conference on Software Maintenance (ICSM).* IEEE, September 2012.

[8] Fouad Nasser A Al Omran and Christoph Treude. 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, May 2017.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[10] Jeff Reback, Wes McKinney, jbrockmendel, Joris Van den Bossche, Tom Augspurger, Phillip Cloud, gfyoung, Sinhrks, Simon Hawkins, Matthew Roeschke, Adam Klein, Terji Petersen, Jeff Tratner, Chang She, William Ayd, Shahar Naveh, Marc Garcia, Jeremy Schendel, Andy Hayden, Daniel Saxton, Vytautas Jancauskas, Ali McMaster, Pietro Battiston, Skipper Seabold, chris b1, h vetinari, Kaiqi Dong, Stephan Hoyer, Wouter Overmeire, and Marco Gorelli. pandas-dev/pandas: Pandas 1.1.4, October 2020.

[11] Henry Tang and Sarah Nadi. Can we use stack overflow as a source of explainable bug-fix data?, 2020.

[12] Christoph Treude and Martin P. Robillard. Augmenting API documentation with insights from Stack Overflow. In *Proceedings of the 38th International Conference on Software Engineering - ICSE '16.* ACM Press, 2016.

[13] Gias Uddin and Martin P. Robillard. How API documentation fails. *IEEE Software*, 32(4):68–75, July 2015.

[14] Di Yang, Aftab Hussain, and Cristina Videira Lopes. From query to usable code. In *Proceedings of the 13th International Workshop on Mining Software Repositories - MSR '16.* ACM Press, 2016.

[15] Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. Learning to mine aligned code and natural language pairs from Stack Overflow, 2018.