RIT

Dynamically Generating Stopword Lists from Software Engineering Artifacts.

By

Shimon Johnson

A Capstone Report Submitted in Partial Fulfillment of the Requirements for the
Degree of
Master of Science in Software Engineering

Department of Software Engineering
Golisano College of Computing & Information Sciences

Rochester Institute of Technology
Rochester, NY
May 6, 2020

Committee Approval:

Dr. Christian Newman

Date

Capstone Advisor
Assistant Professor
Department of Software Engineering

Dr. J. Scott Hawker

Date

Associate Professor
Graduate Program Director
Department of Software Engineering

Dynamically Generating Stopword Lists from Software Engineering Artifacts.

Shimon Johnson
Department of Software Engineering
GCCIS
Rochester Institute of Technology
Rochester, New York.
sj2378@rit.edu

Advisor :- Dr. Christian Newman

***Abstract*— Working with text data is often challenging, various Machine learning approaches like Text classification often require Preprocessed data. Preprocessing is a step in the data mining pipeline which is used to transform raw data into a useful and efficient format. Essentially it helps clean and prepare data before predictive models are developed. The research entailed within this Capstone report primarily focuses on this aspect and describes a tool created from scratch that provides custom stopword lists from software engineering artifacts containing text data, in an attempt to achieve a better understanding of how words within text data impact machine learning approaches by allowing us to study the effects of different heuristics for determining stopwords and how they improve or degrade the results of ML and other data processing approaches, effectively allowing optimizations to data preprocessing for these approaches. Concepts utilized within this tool such as TF-IDF and POS Tagging offer a great start in this direction and are the crux of this tool & research. This report also discusses these mentioned concepts, the adopted research methodology, implementation, evaluation of results and possible future work.**

Keywords— Natural Language Processing; TF-IDF; POS Tagging; Software Engineering; Stopword; Detection

I. INTRODUCTION

Data preprocessing is a crucial step to ensure effective outputs from a Machine learning-based approaches. It a known fact that higher quality of the input provided to these approaches often produces high quality outputs and results. Most natural language text data consists of certain words in the presence of other more meaningful words known as Stopwords, these words add absolutely no meaning or semantic value and hence don't need to be considered at all, ergo fluff which can be eliminated altogether.

Stopword removal is definitely not applicable towards all ML applications that utilize natural language text data. Applications such as sentiment analysis for instance wouldn't positively benefit from stopword removal, in fact stopword removal could cause inaccurate and misleading results from such models and approaches. That being said, ML Applications like Text classification, Spam filtering, Caption generation and Information retrieval could positively be impacted by removing stopwords and examining this effect is the long term goal of this research undertaking. As a stepping stone in this direction, it was important to understand certain concepts and synthesize them into an easy to use tool capable of producing stopword lists from user provided input text data, which in case of this research undertaking was Software engineering artifacts.

Software engineering artifacts were chosen in particular since no previous research or experimentations were carried out on them and therefore piqued interest due to its novelty in the domain. Examining the effects of stopword removal on such documents when provided to different ML approaches offer new avenues in exploratory research and served as the main motivation behind this research undertaking.

II. RELATED WORKS

Raulji et al. [1] with their study presented an algorithm to automatically generate stopwords for Sanskrit Language; Their algorithm was based on the frequency of words and the ease of implementation. They presented a list of 75 generic stopwords from almost seventy-six thousand words. The approach they adopted consisted of tokenizing a stream of data delimited by spaces and newline character then followed by initializing a 'pivot' and 'index' word to carry out comparisons, with each detected match, the stopword counter was incremented and the process was carried on until all index words were compared with the pivot. With all the comparisons completed, frequency was calculated based on the number of occurrences vs total number of words that were present in the data stream. The obtained frequency percentage was compared to a threshold value and was considered as a stopword only if it exceeded the set threshold. The pivot was then set to another word in the stream, and the process repeated itself until the pivot reached the end of the stream. Threshold value of 0.25% was set and a total of 190 stopwords were obtained, inclusive of domain-specific nouns which were then manually removed or eliminated leaving 64 stopwords to which 11 words were manually added totalling to 75 stopwords. Although they claimed to have an automated approach, Manual intervention was necessary thereby hurting their claims and also the algorithm was developed to work specifically with word forms that were space delimited and hence including a word splitter to split fused words in Sanskrit could greatly improve their results like they mentioned.

Ayral et al. [2] proposed an Automated domain specific stop word generation method for Natural Language text classification; they also implemented a Bayesian natural language classifier on webpages in order to test the generated stop words. Their entire approach was carried out on the PASCAL Ontology Dataset, and the list generation included calculating cross-document and cross topic differential specificity values for each word present within the corpus. The proposed approach in their study involved considering for each unique word in the vocabulary of the corpus, the number of unique topics containing that word and the number of unique documents containing the word in consideration. They calculated topic coverage as a ratio of the number of topics containing the word in consideration over the total number of topics, similarly they calculated Document coverage as the ratio of the number of documents containing considered word over the total number of documents, having sorted the vocabulary of the corpus according to these ratios, words above a certain threshold could be chosen to be considered as stopwords. They employed a Bayesian Maximum A Posteriori (MAP) Estimation on a bag of words. The results they obtained proved that document coverage rank and topic coverage rank of the words followed Zipf's law. They presented two lists of stopwords they generated by first sorting number of unique documents covered for the first list and then sorting according to the number of topics covered for the second list. They compared their results with a reference list containing stopwords from the Oxford English corpus published oxford online and the lists contained words like 'home' and 'site' and didn't contain pronouns like 'I','You','He' which were present in the reference listing, therefore, proving that their approach was domain specific.

Schofield et al. [3] through their study, focus on the impacts of stop word removal on topic models by analyzing the consequences of removing stopwords for topic modeling in terms of model fit, coherence and utility. They consider 3 configurations; the first bring models trained and presented with stopwords intact, the second being models with stopwords removed before training and finally the third being models with stopwords removed after training. They make use of a preset list of stopwords containing roughly around 500 stopword types to perform the removal. They hypothesize that A) 'Stopwords harm inference' B) 'Stopwords have no effect on inference' and finally C) 'Stopwords improve inference'. In order to study the effects of the removal of stopwords on the topic quality and keyword generation, topic models are evaluated as Language models and document summarization tools. By studying the mutual information between documents and topics, the MI before and after removal of stopwords can be measured to notice the effect of such a step. They perform their experimentation on two different corpora, a corpus of United States State of the Union (SOTU) addresses from 1790 to 2009 split into paragraphs and a 1% sample of the New York Times Annotated corpus (Sandhaus, 2008), spanning articles from 1987 to 2007 and split into 500-word segments to handle overly-long articles. The results they find suggest that removing stopwords after training is pretty effective just as removing before training also that generating a corpus-specific stopword list to perform the removal doesn't provide much utility. Their study made use of a predefined list and had nothing to do with actually generating stopword lists.

III. Research Methodology

On the backend, two very important concepts form the crux of this tool and research, TF-IDF and POS Tagging. Term Frequency-Inverse Document Frequency / TF-IDF is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. Essentially it works by increasing proportionally to the number of times a word appears in a document, but is offset by the number of documents that contain the word.
It is calculated by multiplying two metrics, TF and IDF.
TF represents the term frequency of a word in a document and IDF represents Inverse document frequency of the word across a set of documents. By calculating both these metrics and multiplying them, a numerical score can be assigned to each word.

The reason why TF-IDF serves as an important starting point is due to the fact that ML approaches with natural language often make use of algorithms that typically deal with numbers and that natural language comprises of text. Hence there exists a need to transform such text into a format that can be understood by these algorithms, often called Text vectorization.

POS Tagging is the process of marking up a word in a text as corresponding to a particular part of speech based on both its definition and its context. POS tagging serves as an extension to TF-IDF within this research since contextually determining stopwords was something that was intended to be built right into the tool. It helps ascertain words that are definitely stopwords and those that should be considered as stopwords. As an instance of this, Determiners such as 'a', 'an', 'the', among others simply don't convey any semantic value within a sentence and therefore are definitely stopwords. POS Tagging aids in understanding words like these by identifying their tags and thereby understand their semantic value within a sentence.

Although TF-IDF by virtue assigns a lower score to words that appear commonly across documents within the corpus and a higher score to words that are not very commonly found across documents within the corpus, it would be commonplace to assume that stopwords would consistently obtain a lower score due to their presence in almost all documents within the corpus. This assumption is where TF-IDF as a standalone approach fails and can be better supported by natural language processing approaches such as POS Tagging. This tool and research undertaking embark on synthesizing these two approaches in an effort to dynamically produce stopword lists from input documents provided by the user, which in this case are software engineering artifacts. (API Documentation produced by Javadoc)

Kindly refer to the Readme.MD file available on Github for details about libraries and modules used to build this tool and the steps to run it as well.

The research starts by building a corpus of software engineering specific text documents, to that end, API documentation produced by Javadoc was considered. This documentation was obtained by scraping HTML pages from Oracle's website using the web extractor/ text scraper and placed within the 'Input_files' folder, Alternatively a user could also simply drop text documents directly within the 'input_files' folder, if no web scraping is needed and the text files are directly available.

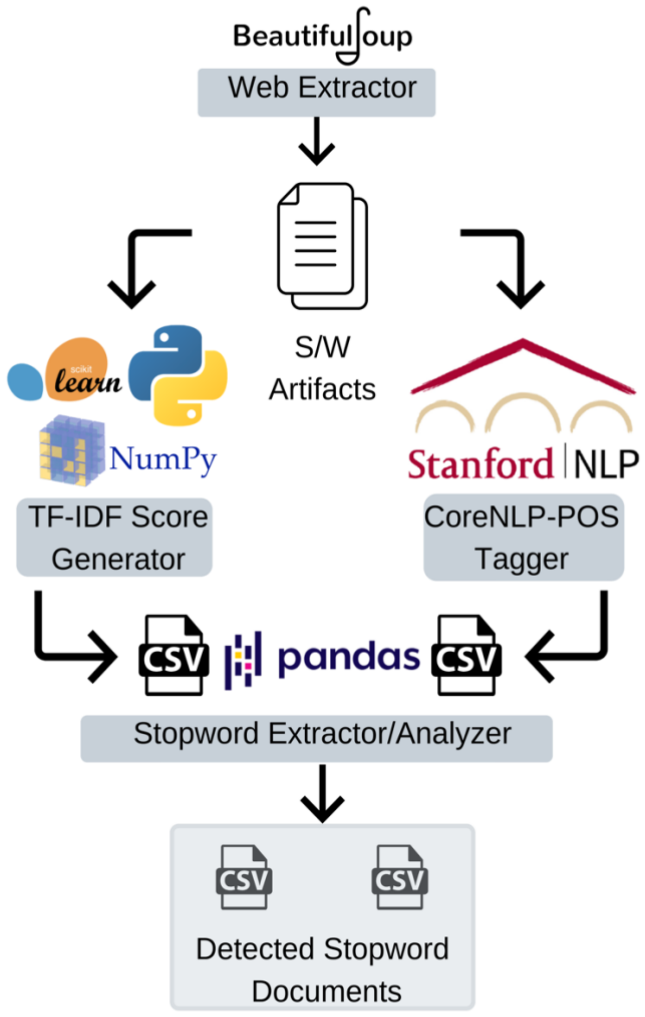

Fig 1:- Tool Implementation & Tech Stack

Contents of the 'Input_files' folder are then processed by the TF-IDF Score Generator. The score generator is based off scikit-learn's tfidfvectorizer, which very effectively calculates a score for each word present within the document.

Tfidfvectorizer's default parameters were modified to accurately reflect scores in the desired format and range, some parameters were not considered at all, this was done in an attempt to obtain results best suited for this study. Other libraries such as Numpy and Pandas are also used in tandem with scikit-learn to carry out data manipulation, effectively generating CSV files containing scores for each document within the 'Input_files' folder. These scored CSV files are automatically placed into a different folder called 'Scored_files' and a threshold was determined by performing manual analysis of the obtained results.

Determining a threshold score was a significant aspect of this research undertaking and was carried out by manually analyzing the results of this tool. Each scored file was filtered by considering this threshold and the results of that filtering were then saved onto new CSV files within the 'Filtered_files' folder. Determination of this threshold would be further outlined in the next section of this report.

Once each document within the corpus is scored, documents within the corpus are then subjected to POS Tagging, carried out by the Stanford CoreNLP POS Tagger which makes use of the Penn Treebank Tag set and contextually assigns each word in each document with a POS (Part of Speech) tag followed by saving the tagged txt files within the 'POS_output' folder.

Files generated by both the Score generator and POS Tagger are now collectively analyzed by the Stopword Extractor/Analyzer, which checks for a word's score and its corresponding POS tag before flagging it as a stopword, this process is repeated until all words within all corresponding files are checked. Details on the POS tags which enable the tool to flag a word as a stopword would be discussed in the evaluation section of this report.

Flagged words from each original input files and corresponding intermediary files are then compiled into a final CSV file within the 'Required_results' folder, that is then presented to the user and essentially contains a custom list of stopwords for each specific input file that the user added into the 'Input_files' folder at the beginning of the process.

IV. Evaluation

The tool was subjected to five software engineering artifacts which were API documentation obtained from oracle's website using a web scraper and were then converted into text files and placed into the 'Input_files' folder.
Each of these documents were based on different domains within oracle's Java documentation library to ensure that, 1. Obtained results wouldn't be biased on any single domain in specific and 2. Diversity within the obtained results.

Files from the score generator were manually analyzed on excel by sorting them from high to low scores and then applying a normal distribution to them to examine how closely words are gathered around the mean of the dataset. This distribution and corresponding scores were then plotted on a bell curve to help visualize words within the documents present in the dataset. This process was carried out for all input documents.

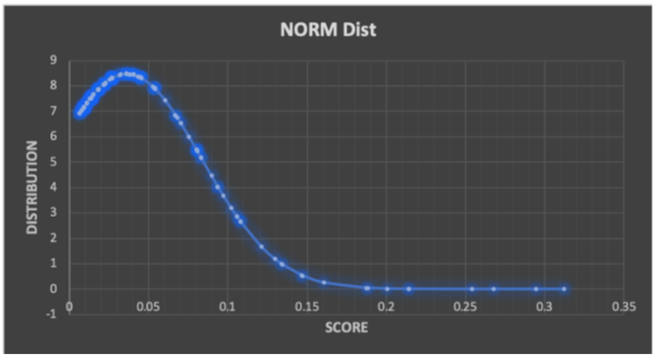

Fig 2:- The Distribution Plot for one I/P file

An initial fact that most stopwords are Determiners (DT) shaped the definition of a stopword within this approach. I began with that fact as my initial starting point and kept manually evaluating the tool's results. With the initial definition of a stopword being any word that had a 'DT' /

determiner tag, manual analysis of the results showed that there were still words within the document that could be considered as a stopword and that's where the optimization of the tool began. I came to a realization that optimization of the tool was necessary and began manually annotating words that showed potential to be a stopword, from this I discovered that words with tags such as 'IN', 'CC', 'PRP', 'WDT', 'PRP$', 'WP', 'WP$', 'RP', 'TO', 'PDT', 'WRB' and 'CD' could also be considered as stopword since even they semantically didn't add any value to any of the documents. Thus, a stopword's definition within this approach now expanded to add in not only Determiners but also Prepositions, Coordinating conjunctions, Personal pronouns, Wh-determiners, Possessive pronouns, Wh-pronouns, Possessive Wh-pronouns, Particles, Predeterminers, Wh-adverbs, Cardinal numbers and the word 'to'.

By broadening the definition of a stopword within this approach, a significant improvement in the results were observed, especially when compared to the results of the initial analysis that was carried out.

V. Threshold Determination

As mentioned earlier in this report, Determining a threshold score was a significant aspect of this research undertaking so as to know which words should be considered stopwords.
It was carried out by manually analyzing the results of this tool on Microsoft Excel. To this end, multiple values within the normal distribution were considered. Mean, Median, Percentiles (25th,50th and 75th) and standard deviations(+1SD,+2SD) were candidates under consideration.

By manually examining the number of stopwords that were determined when scores were filtered below the potential thresholds I particularly looked for a maximization in the stopword detection front. After conducting this manual examination across all documents in the corpus for all candidate thresholds, it was observed that a maximization of stopwords was achieved when the threshold was set to the positive second standard deviation (+2SD) and thus was chosen as the threshold below which all words obtaining a score would be checked for their POS tags by the Stopword Extractor/Analyzer before flagging them as stopwords.

VI. Results

In order to test the validity of the results the tool gave, I compared it with scikit-learn's list of English stopwords which were placed inside a CSV file called 'target'. It was a way to see how valid were the lists of stopwords that the tool presented as output when compared to the ground truth, in this case scikit's list.
To that end, a python script was written that would automatically carry out this comparison and visualize the results to make it easy to understand.

From the obtained visualization, 19.2% of the words within scikit's list of stopwords were correctly detected as stopwords by the tool. 2.5% of the tool detected words were uncommon between scikit's list and required_results. It was also observed that 78.3% of words that were present in scikit's list of stopwords were either completely absent in the Input data provided to the tool or had different POS tags that were considered and hence were not detected by the tool, thereby highlighting the importance of building a very diverse corpus of documents.

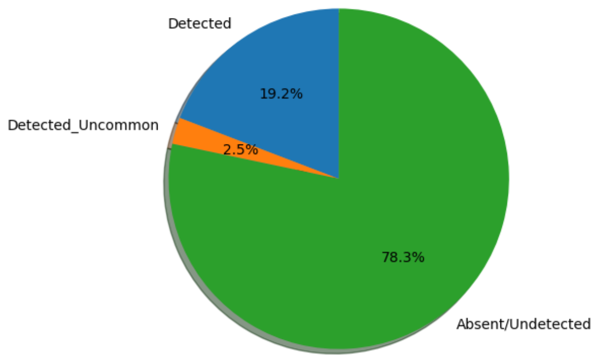

Fig 3:- Visualization of Comparative Analysis

VII. Future Work

Observations carried out during manual analysis point out to the fact that some words within documents aren't split in the best manner, some since they are developer defined identifier names and some since they aren't present with delimiters. Scikit-learn's tfidfvectorizer doesn't work appropriately within the context of software engineering artifacts and therefore embedding a custom splitter like spiral which is capable of performing various different types of splits would be the next logical expansion to this study and tool. Another observation that can be drawn directly from the results is need to build a corpus of software engineering related artifacts and documents due to the lack of any preexisting corpus of such documents. Results from the tool can be optimized and further refined by considering other NLP approaches such as stemming or lemmatization or Morphological segmentation in addition to TF-IDF and POS Tagging. This tool could also greatly benefit from using Deep Learning Neural networks and that is another avenue one could pursue from this initial tool and research undertaking.

VIII. References

[1] J. K. Raulji and J. R. Saini, "Generating Stopword List for Sanskrit Language," 2017 IEEE 7th International Advance Computing Conference (IACC), Hyderabad, 2017, pp. 799-802.
doi:10.1109/IACC.2017.0164
URL:http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7976898&isnumber=7976734

[2] H. Ayral and S. Yavuz, "An automated domain specific stop word generation method for natural language text classification," 2011 International Symposium on Innovations in Intelligent Systems and Applications,Istanbul,2011,pp.500503.doi:10.1109/INISTA.2011.5946149
URL:http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5946149&isnumber=5946042

[3] A. Schofield, M.Magnusson, D.Mimno, "Pulling Out the Stops: Rethinking Stopword Removal for Topic Models" https://www.aclweb.org/anthology/E17-206