Identifying EOL Software

Neal Damireddy[1] and Clayton Greenberg[#]

[1]Foothill High School
[#]Advisor

ABSTRACT

Throughout our project, we have been exploring ways to identify EOL (End of Life) and malicious websites using python. With this software, we can help people stay away from these types of websites. There are lots of cybersecurity problems in the world, but we wanted to address the EOL problem because it is something that is within the realm of what we are learning (language recognition), and it has a practical use in the world. We used multiple different commands to identify certain "keywords," with the intention of getting the highest possible accuracy percentage. We also prioritized the elimination of false negatives over false positives. Creating a new variable with the keywords, we were able to correctly predict 99% of the EOL websites using the data given to us. We were able to conclude that mixing the 2 most important data columns and creating one variable to determine both of the variables is the best way to go about creating a variable that gives you the most.

Introduction

Our research problem is to identify the EOL (End of Life) websites and how we can identify them. Using this software, we are able to help people who aren't able to identify these problems themselves. We were able to add this software to a bigger cybersecurity software company called RiskSense, Inc. We used supervised learning for our data set. Along with that, we used classification to separate our data into 2 decisions (EOL or not EOL). Our data was categorial, and it was also language data, made mostly of words.

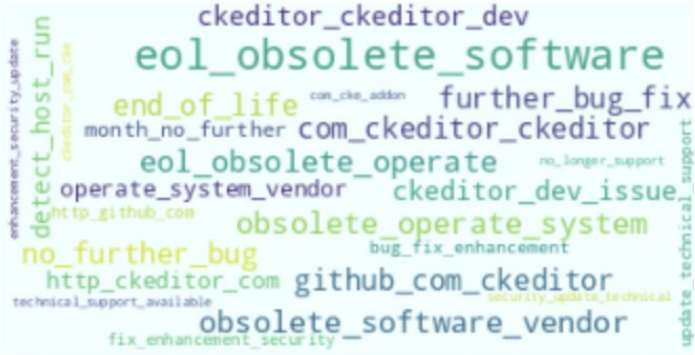
Background

We were able to take inspiration from a previous set of code that was provided to us to guide our way into how we should create our project. In this sample, like us, they first created a variable for training called df_train. They then downloaded the nltk package and used it to cleanse out their data, which helped the computer more easily analyze the data. They then deleted the duplicates in their dataset and located all the data points without a title or description. They then replaced all of these blank titles with a space, or blank. Unlike us, they used 5 words for their "new" description, while we used 3 words, also known as a trigram. They then imported ngrams to classify the data into either "true" or "false" vectors as a way of simplifying the process. Lastly, they made their BOW (Bag of Words) model using the pentagrams instead of trigrams. Along with this, they also used logistic regression with cross validation to train the dataset.

Dataset

This dataset is made up of descriptions for updates for various software packages. Somebody labeled each of these updates as end of life or not. Included are 1522 separate updates. Each update includes a title, description, whether the update is EOL or not, and the plugin type. The title first tells you what software package this update is working on and what section of the software is being updated (Ex. Security). There is an average of 7.4 words per title throughout

the data set. The description gives an in-depth explanation of what the update is actually doing, and why this update is necessary. There are on average 68.2 words per description. In our data set, about 837 of the points are classified as EOL, while about 685 are classified as Non- Eol. To get the most accurate description for the computer to read, we created a clean_description variable which combined the "title" column and the "descriptions" column and then cleaned out the data. We also stripped the HTML code from the areas that had the code in it.



Our data set type is language, and this word cloud here represents how common a word is in the data set. We split our data 80% to 20%, meaning we used 80% of our data to train the program, and 20% of the data to test how accurate the program was.

Methodology/Models

We tried to use word2vec and a simple train command, but neither produced great results. A word2vec model is a model in which you train the program to learn certain words, and group together those that have a similar meaning. For example, the words "EOL" and "Obsolete" could be grouped together because they both have the same connotation, meaning the website or software is "dead" in a way. We ended up creating our own variable to identify whether something was EOL or not. The pros of this approach was that it is very accurate when you have a certain word such as "eol" or "obsolete." We aren't sure how this code will perform under a different data set, especially if there are no such words. First, we cleansed out the data set using a package called nltk which included stopwords, a premade list of common words (the, it, a) to remove from the dataset, punkt for tokenization, averaged perceptron tagger for speech tagging, lemmatizer for removing the -ing at the end of a sentence (writing -> write), and own-1.4 for the words that are in different languages. We then created trigrams to make the data more easily readable. There were a total of 39,962 trigrams, or groups of 3 words in our data set after we had completely cleansed out the "title" and "description" column of the data set. Our data is split between a 80% train and 20% test; in our test data, we were able to achieve as 99% correct prediction rate. We then used a BOW(Bag of Words) model with the trigrams in the new clean_descriptions variable. The way a BOW model works is it takes some words, whether it is from a sentence or a paragraph, is represented as a "bag" of words. To keep it simple, the grammar and word order don't matter. The trigrams used in the run of the program that we used could be completely different compared to another run that we had because the trigrams are randomly generated. We compared the correlation between the trigrams against each other, a whopping ($39{,}926^2$) calculations, and the trigrams that had too much correlation with each other were removed; and the trigrams that had no correlation with each other were also removed.

Results and Discussion

The goal of our project was to create a program that would be able to take a piece of software and determine if it is EOL or not, or malicious or not. I had the opportunity to create this software for a bigger company which was then added to their overall cybersecurity software. This project really interested me because it was an opportunity to help people and learn about how you create a code for cybersecurity, and all the checks you need to have to make sure there are no flaws in your code, and it can't be easily exploited. The results of this project were very pleasing, and I was able to achieve the goal that I had set for myself, which was to create a working and efficient program which would have actual real-world use. Like I said before, we created a new variable to help the computer bunch up all the data and identify important saying that would tell you if the software was EOL or not. We tried using a word2vec model, but after some time and attempts to tweak the code, we were only able to yield a 55% correct prediction rate. We then looked in another direction, in the form of a Bag of Words model. This model did substantially better, and after splitting the data into trigrams, we were able to achieve a 99% correct prediction rate. Some flaws that this code may have is it might not be reproducible. For example, the data set we used clearly outlined whether it was EOL or not in the title and descriptions. I wonder how the program would perform under some more obscure software, where it might not be clearly said whether the software is obsolete, EOl, etc. I believe that our program wouldn't perform as well on more obscure data because it is very easy to identify something that is EOL if it is clearly said in the title.

Conclusion

Throughout our project, our goal was to create a code that could be used for a bigger cybersecurity company. We used lots of different approaches but creating the new variable with the keywords turned out to be the best way. I think our model performed well in this dataset because the keywords were easy to identify, but I'm not sure how it would perform in a real-world situation. The next steps to take would be to try out the data on a different dataset and see how it performs. The Bag of Words model easily was the best working approach to this problem, but it leaves me wondering if there were any tweaks, I could have made to the dataset on the word2vec model that would yield better results, and maybe potentially more repeatable results in other software that may be more obscure.

Acknowledgements

I would like to thank and acknowledge Clayton Greenberg, who was my mentor and helped me heavily throughout the project. I would also like to acknowledge RiskSense, Inc., who provided me with the data to work on this project.

References

Takata, Yuta, et al. "Identifying evasive code in malicious websites by analyzing redirection differences." *IEICE Transactions on Information and Systems* 101.11 (2018): 2600-2611.

Khan, Hafiz Mohammd Junaid, et al. "Identifying generic features for malicious url detection system." *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*. IEEE, 2019.