Question 3 and 4: Report Charlie Galvin

In all of my research , as I am doing NLP (a known, new research area in computer science that is still evolving), I wanted to look into reliable algorithms that I could take inspiration from. Whilst algorithms such as BERT and Jaccard's looked interesting, Bag-of-Words (BoWs) and Word2Vec seemed to be two very popular algorithms that also seemed relatively easy to implement. I had read in many articles the following:

- BoWs: This is beneficial for document classification. It investigates the frequency of the words in each document in the corpus and uses it to train the model for making comparisons
- Word2Vec: This investigates the distribution from each word to one another and thus works on semantic relations between words, which BoW's cannot do as it focuses more on the lexical than the semantic.

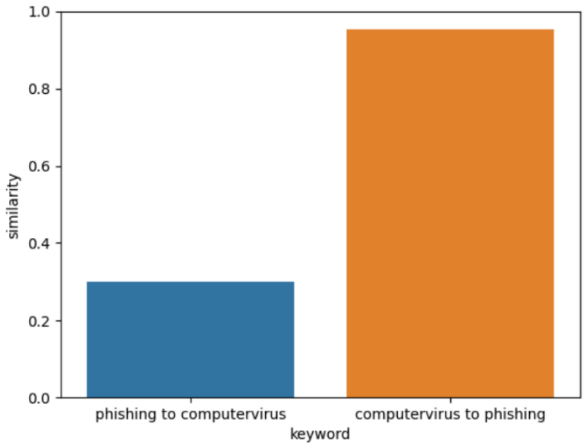
I feel as though both are very important when we consider semantic distance; a measure at how "close" in meaning one article is to another as this is subjective. An article talking about "Loving Dogs" and another talking about "Hating Dogs" may be lexically similar, but also semantically similar as both are talking about dogs. In contradiction, some could say that they are not semantically distant as they have completely differing opinions. This negotiation on the subjective is why I have chosen to use both models in my algorithm. To consider both document classification and semantic closeness in comparisons to evaluate the semantic distance. Lastly, when finding "relevant articles" in my data collection, I created a relevancy list to address most frequent to least frequent articles with the number of times the "Keyword" appeared in the article. The more relevant an article is when being semantically compared to a different keywords corpus, the more it's "score" should contribute to the overall distance semantic. I have thus involved this "relevancy list" in my q3 and will speak about how this contributes later.

Description of the algorithm:

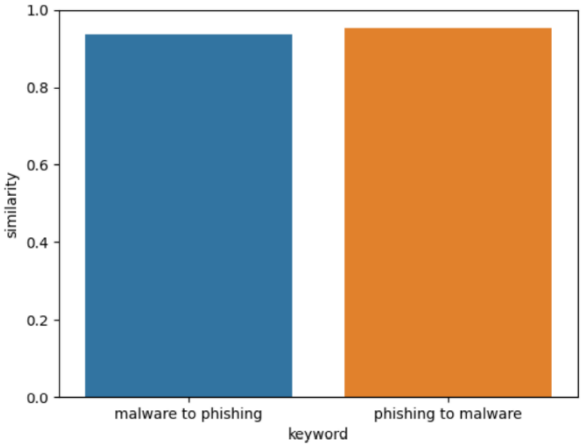
My algorithm is broken into 3 parts: Word2Vec, BoW's, main body.

For Word2Vec similarities, my algorithm opens the text file "semantic_corpus.txt", which contains non-cleaned data and removes all punctuation from it. It then tokenises each word in each sentence, to convert into a Word2Vec model using genism.models.Word2Vec. I set the min_count to 1 as otherwise it got rid of keywords that appeared too few times in the corpus, including other keywords I wanted to compare.

In pre-processing, for any word in the semantic corpus that was either a "necessary word" or "influencing word" related to the corpus (see question 2 python), I replaced the word with the keyword itself. I made this decision to increase the chances of having enough results to build a relationship in word2vec, with the assumption that when we considered the "relevant articles" as those that contained the most "necessary and influencing words", we were thinking they those words were semantically identical to the keyword itself. With this assumption in mind replacing each necessary and influencing word with the keyword seemed logical. We then calculate the similarity using word2vec.we.similarity. We save the "semantic relationships" matrix in the file *output_similarity.xlsx*. Using sns.barplot, we then show the semantic similarities between every Keyword in the folder "*files"*, as well as save each one in the pdf *output.pdf*.

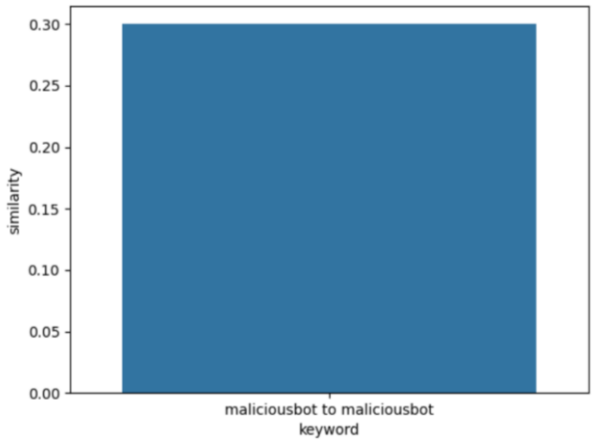


(a) Relationship between phishing and comptervirus



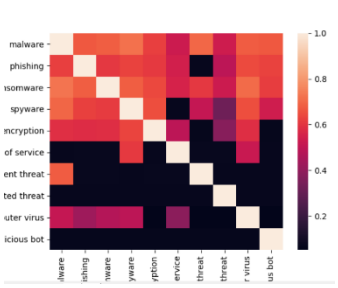
(b) Relationship between phishing and malware

It is clear from this image that while malware and phishing are very semantically similar, both with a very high meaning, from (a), we can see that articles that talk about computer virus often talk about phishing but not the other way round, and thus it could be assumed that phishing can cause computer viruses but not the other way round. This assessment of causation and similarity is why I visualised this data.

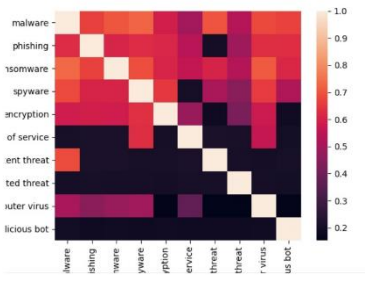


I'm not sure what this data means though as our heatmap at the end gives a score of 1 to the two words, but using this data it should only give 0.3. I have not been able to find a reason for this.

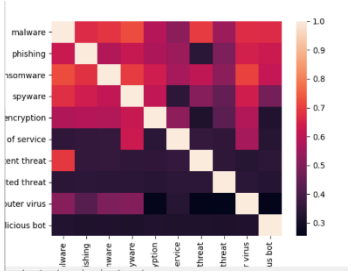
Furthermore, I have added a variable threshold in my algorithm so that if the comparison keyword is not in the keyword's corpus, then some punishment must be met as it shows either that our corpus is not big enough or that they are semantically distant. The 3 heatmaps that I used with sns, show the results of changing the threshold:



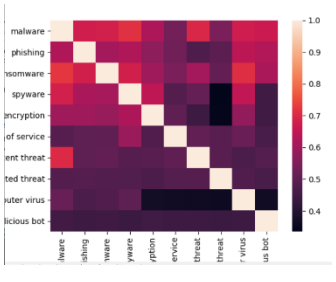
a) Threshold set at 0.1



b) Threshold set at 0.3



c) Threshold set at 0.5



d) Threshold set at 0.7

Even though I tried to increase the size of the corpus by adding the search for "bot" into the search bar in BBC NEWS, and include only websites that contained at least one of "malicious" or "harm" in my question 2. This increased the size from 8 to 17. I also looked at FOX, CNN, CBC, The Daily Mail and The Guardian, but none had a convenient search bar that also allowed me to find links to the relevant websites easily. Because we have limited corpus's, I set the threshold at 0.3.

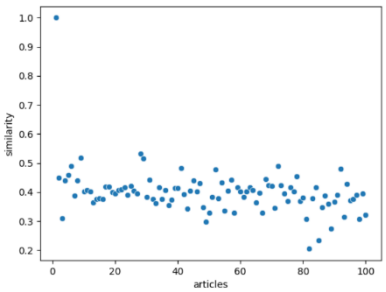
The second part of my algorithm looks at "Bag of Words". For each word, we open "collection1.txt", a file that has been cleaned (see python problem 2). In pre-processing, we remove punctuation and then for every file we write to "collection1.txt" we add a full stop after. This means when we tokenize each sentence, each article is a token. We then tokenize each word in each "sentence", and for visualisation purposes we have made a data frame and written it to "*Bag of words.xlsx"* in each keyword directory.

We then use genism.similarities.Similarity(directory for keyword, corpus of keyword) to build a similarities list called *sims* for each file in the keywords directory. (E.g. a list that says how similar each file is to the corpus).

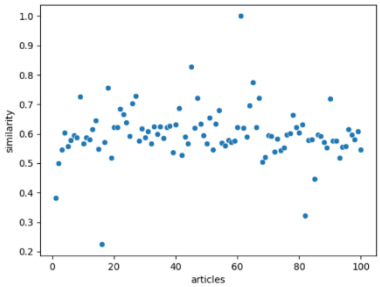
In the main section, for each keyword1, we do a keyword comparison with keyword2, in the following way:

- For each comparison keyword2, we do Word2Vec to get semantic similarity for the relationship between keyword1 and keyword2, using keyword1's corpus. We also open *"relevancy list.txt"* for keyword2.
- For each individual article document in keyword2's directory, we tokenize each sentence and then each word in each sentence. We then compare it's similarity to keyword1's corpus using sims, and return it's maximum value. I chose to return the maximum value as I though if an article is in the corpus already, for example file "*malware000*" being compared to malware1's corpus, "*collection1.txt*" in the malware folder, but this isn't similar to any other article in the corpus so it's average would be like 0.01 (similar to first 1 so 1, not similar to other 99), it should still return a value of 1 for it's similarity as it *is in* the corpus.

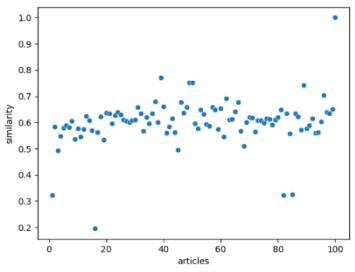
  I have used sns.scatterplot to show how similar each article for a keyword is to **only** that keyword's corpus. This is to visualise how relevant the article is in the corpus and how similar it is to all other articles in the keyword's corpus. I did not do this for every result as instead of showing 1000 scatter plots (100 for each keyword), it would have been 10000 which I thought was too much and not useful.



Malware corpus: Article 001 compared with all others in the malware corpus



Malware corpus: Article 060 compared with all others in malware corpus



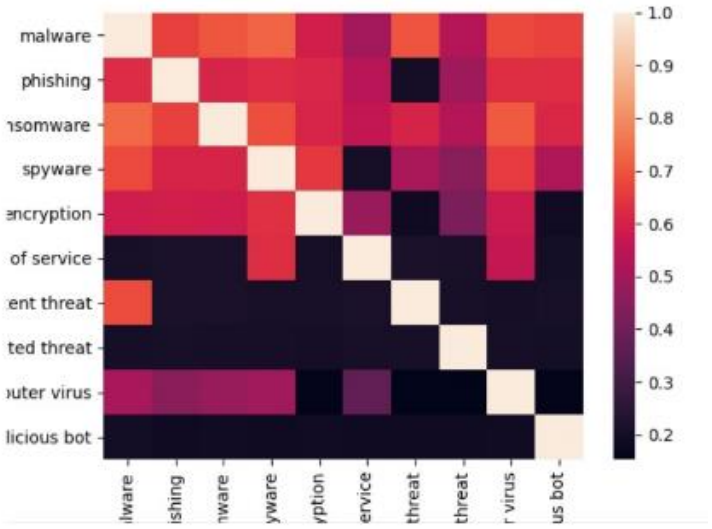
Malware corpus: Article 099 compared with all others in malware corpus

- Using the keyword's relevancy list, if keyword2's article is considered "more important", this should influence the effect it has on the overall distance more. We represent "overall semantic distance" with the variable *summation.* The overall formula is:

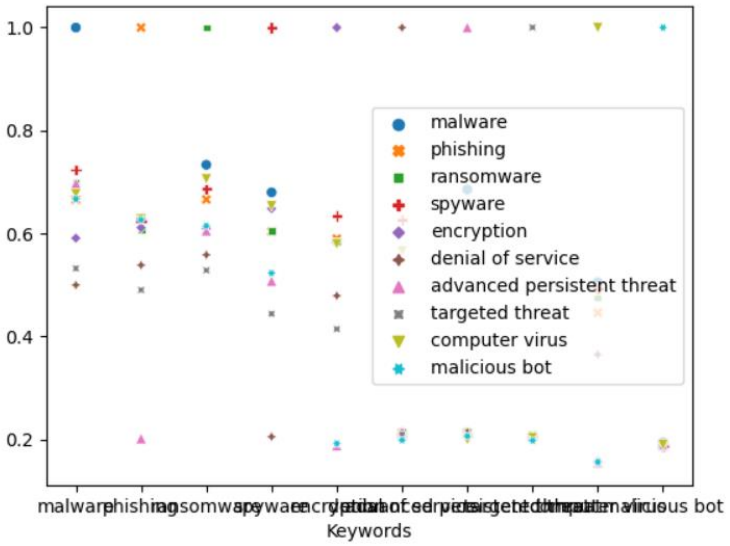
Summation= Individual article comparison of BoW's for keyword1's corpus and keyword2 **X** Article relevancy in keyword2's relevancy list **X** Semantic similarity between keyword1 and keyword2.

Note that since the individual comp and the semantic similarity always give an answer between 0 and 1, and a summation of every value in article relevancy is equal to 1, the result will always full between 0 and 1 making it easy to compare results.

The results of the algorithm can be seen below:



I have used sns.heatmap to show this data. Although have not had time to make it look nice. This is useful for looking at the overall spread of results.



I have used sns.boxplot visualise this data. Although have not had time to make it look nice. This is useful for individually comparing how far each keyword is with a specific keyword.

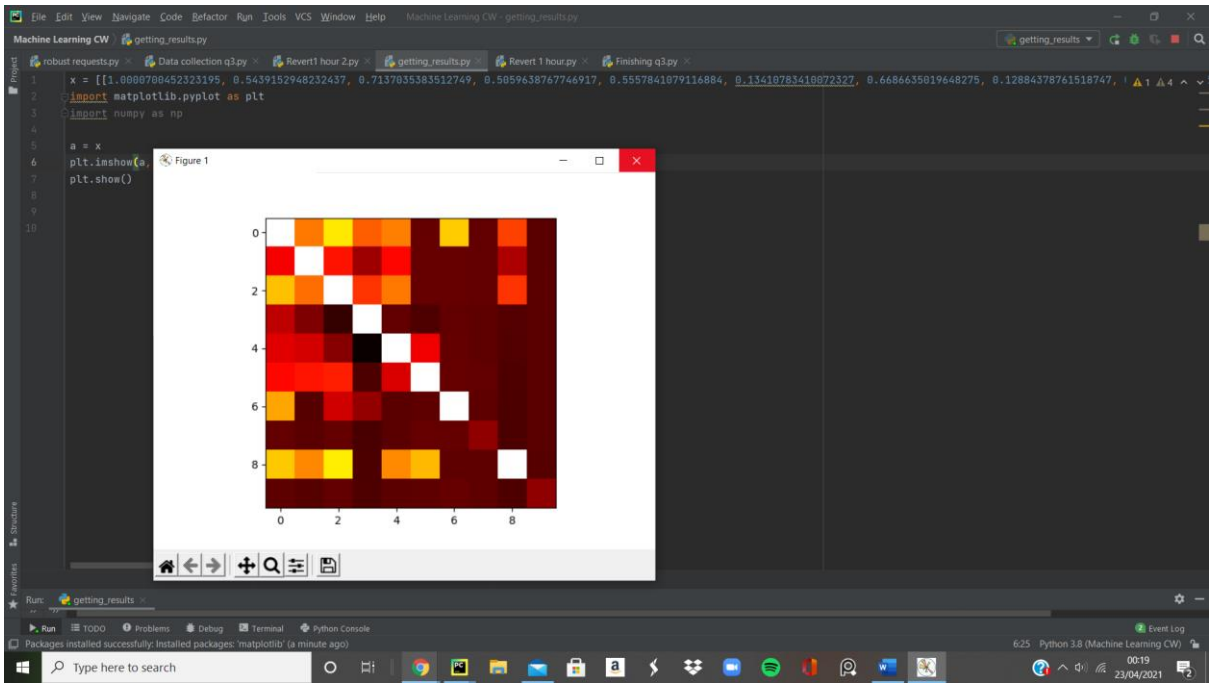
Bibliography:

https://stats.stackexchange.com/questions/312206/can-i-apply-word2vec-to-find-document-similarity

This website kind of states not to use word2vec with IF-TDF or BoW as this is already incorporated in word2vec.

https://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.YIHwXuhKhPY

Helped me realise YOU need to set the min_count to 1 or 2 otherwise it sets it higher and your comparisons of other keywords are lost. I think it also helped me justify my choice in the corpus.



Bibliography

Remove punctuation from a string:

https://www.geeksforgeeks.org/python-remove-punctuation-from-string/

https://www.youtube.com/watch?app=desktop&v=hxU3HLX1W1k

https://stackabuse.com/python-for-nlp-creating-bag-of-words-model-from-scratch/

Understanding the definition of semantic similarity: https://medium.com/@adriensieg/text-similarities-da019229c894

What it taught me:

- Text similarity is determined by surface closeness (lexical similarity) and meaning (semantic similarity). This taught me that comparing frequencies of words although could be ideal if two articles were similar in very extreme cases, E.g. Is Mount Vesuvius still dormant? & "Eruption of Mount Vesuvius in 1944", in many cases this will not help at all such as "Why we should all love dogs" and "Why we should all not love dogs". Both will have very similar word frequencies when compared with one another but the latter is semantically very distant, whilst the former is semantically close. It then lists several ideas on how to do this, two of which I have merged.
    a) Bag of words with/without TF + Cosine Similarity: Which it does not recommend
    b) Word2Vec + Smooth Inverse Frequency + Cosine Similarity: Which it does recommend
- The benefit of Cosine Similarity and how it is used.

https://www.machinelearningplus.com/nlp/cosine-similarity/

Also helped me understand cosine similarity.

The word embeddings I have used within my model are: Bag of Words(BoW) and Term Frequency-Inverse Document Frequency (TF-IDF).

The reason I have decided to use both, which may just seem to add unwanted complexity, is because I believe, as many other websites such as https://medium.com/@adriensieg/text-similarities-da019229c894, … , state: BoW or TD-IDF is good for classification documents as a whole, whilst Word2Vec is good for identifying contextual content. I think both of these contribute to semantic distance, as if they classify into the same group as a whole, but each sentence disagrees, they are still close to an extent, and if each sentence follows a similar contextual idea but the documents are in different categories they are also going to be close. Thus I have decided to use both methods.

Understanding what Bag-of-Words Model is:

https://machinelearningmastery.com/gentle-introduction-bag-words-model/#:~:text=A%20bag%2Dof%2Dwords%20model%2C%20or%20BoW%20for%20short,as%20with%20machine%20learning%20algorithms.&text=A%20bag%2Dof%2Dwords%20is,A%20vocabulary%20of%20known%20words.

This website taught me:

- Bag of words is a way of representing text data that is simple to understand and implement and is good at document classification.
- It is called a "bag" as any information about the order or structure of words in the document is discarded. The model is only concerned with whether knows words occur in the document, not where – They are similar if they have similar content.
- This taught me the process of bag of words to:
    a) Design the vocabulary by splitting the model into individual words

b) Create document vectors. If we can bag all of the "content" into 10 words in the bag, we can mark the presence of words in each "token" as a Boolean value. Yes if it exists, no if it does not. This builds a binary vector for every "token" in the "content".

c) Told me how to pre-process the data, through removing punctuation, frequent words in the English language that won't contribute called "stop words", "lemmatization" and "stemming". I HAVE NOT LOOKED INTO BIGRAMS YET.

- TD-IDF is superior to Bag-of-Words Model as the problem with scoring frequency is that "highly frequent words start to dominate in the document, but may not contain as much "informational content" to the model". An approach of rescaling the frequency of words by how much they appear is called TF-IDF where TF is a "scoring of the frequency of the word in the current document" and IDF is a "scoring of how rare the word is across all documents (the corpus)". This tells me at the end that a limitation of Bag-of-Words is that it discards the "meaning of words in the document (semantics)" but is a "great success on problems like document classification".

Limitations of Bag of Words:

https://www.mygreatlearning.com/blog/bag-of-words/

https://machinelearningmastery.com/gentle-introduction-bag-words-model/

https://wiki.pathmind.com/bagofwords-tf-idf#:~:text=Each%20word's%20TF%2DIDF%20relevance,also%20adds%20up%20to%20one.&text=The%20main%20difference%20is%20that,content%20and%20subsets%20of%20content – the purpose of BoW and Word2Vec

This is used to implement the Bag of Words model:

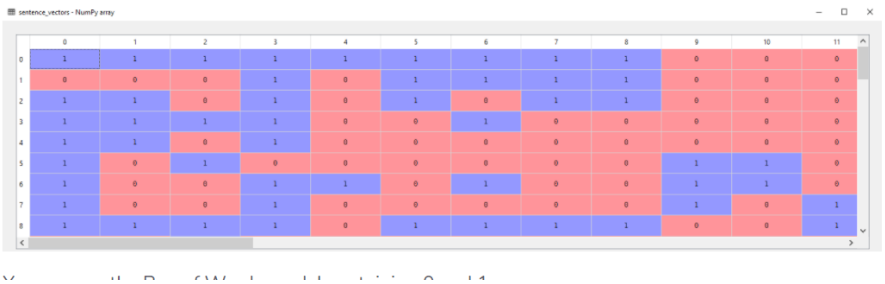
https://stackabuse.com/python-for-nlp-creating-bag-of-words-model-from-scratch/

I used the following steps in my code:

- Step 1: Tokenizing the sentences. In my case this was going to be each articles and I knew that I was going to use nltk.sent_tokenize to do this, so when it came to pre-processing I was going to have to remove all the punctuation and then add a full-stop to the end of the "article text" which I have called "article" before I write the content into a "keyword0**.txt" file.
- Step 2: Create a Dictionary of Word Frequency, where I will have each "word" in the "bag" as the key and the frequency of that "word" as it's value. Originally I just used basic code to do this but found out I could use the genism.dictionary CHECK THIS.
- Step 3: Creating the "Bag of Words" Model. This is to do with point (b) above, making binary vectors for each "token" (which in our case is each article).

- Looking at the example code they give gave me an idea on how to use the corpus, as a way of building a large set of "tokens" that we can use to create the vectors for each article.

- `sentence_vectors = np.asarray(sentence_vectors)`



can potentially use this idea of code to visualise/display the vectors for q4.

https://www.mygreatlearning.com/blog/bag-of-words/

This article gave assistance in both the theory of "Bag of Words", where we pre-process the information by lowering the case of all characters, removing stop words and special characters, converting the words into a "bag" and building a dictionary with their frequencies and then binary vectorizing each token. This website also taught me how to use the CountVectorizer() function to easily implement a Bag of Words program, which allows us to use bigrams and just include the stop words easily. We can then display the DataFrame using pandas, to help us with visualisation.

This article also helps us describe Tf-Idf and how to use it. It also gives us code for Feature Extraction with Tf-Idf vectorizer. – Not sure if using all the malware articles as a corpus might mean that "malware" is not seen as important as it is so popular? – Is your choice of corpus correct?

Understanding nltk :

https://www.geeksforgeeks.org/nlp-how-tokenizing-text-sentence-words-works/ - sent_tokenize, punkt, etc.

https://stackoverflow.com/questions/12703842/how-to-tokenize-natural-english-text-in-an-input-file-in-python

Calculating similarity using Word2Vec:

https://stackoverflow.com/questions/21979970/how-to-use-word2vec-to-calculate-the-similarity-distance-by-giving-2-words

Useful for understanding the theory of Word2vec: https://towardsdatascience.com/word2vec-for-phrases-learning-embeddings-for-more-than-one-word-727b6cf723cf

Pre-processing help:

https://intellica-ai.medium.com/comparison-of-different-word-embeddings-on-text-similarity-a-use-case-in-nlp-e83e08469c1c

This was really useful for telling me about pre-processing the text, extracting the freatures using bag-of-words, TF-IDF and word2vec and then using Cosine similarity. Introduced me to importing stopwords with nltk.corpus, nltk.stem import WordNetLemmatizer. A potential library for Tfidf could be sklearn.feature_extraction.text import TfidfVectorizer. Inseatd of using Wikipedia dataset to

pretrain on, we have our own corpus to work with. - Said about improving the algorithm with Smooth Inverse Frequency.

https://www.geeksforgeeks.org/python-lemmatization-with-nltk/

https://www.geeksforgeeks.org/python-stemming-words-with-nltk/

Understanding what these processes mean:
https://www.datacamp.com/community/tutorials/stemming-lemmatization-python

Understanding what is means to tokenize:
https://www.tutorialspoint.com/python_text_processing/python_tokenization.htm#:~:text=In%20Python%20tokenization%20basically%20refers,in%20programs%20as%20shown%20below.

https://towardsdatascience.com/word2vec-for-phrases-learning-embeddings-for-more-than-one-word-727b6cf723cf

Taught me how Word2Vec worked with "Distribution of words" to gage with semantic distance and that there were many uses for this. This made me decide to use it.

https://stackabuse.com/implementing-word2vec-with-gensim-library-in-python/

This also taught me how to use Word2Vec to compare/look at similarities between words and inspired me to look at the Word2Vec.wv documentation where I later found Word2Vec.wv.similarities.

We may need to not use Tf-Idf as this would mean the IDF value would be 0 as seen here
https://www.baeldung.com/cs/ml-similarities-in-text

Although this may make sense. Malware in an article will have little value in the corpus talking about malware. Thus if malware is brought up in a "phishing" article it will mean nothing. However, I think we don't want this as it should be important so will maybe decide not to do Td-idf.

This helped me understand the definition of a corpus - https://en.wikipedia.org/wiki/Text_corpus

I think removing the stop-words achieves the same thing as td-idf.

https://betterprogramming.pub/introduction-to-gensim-calculating-text-similarity-9e8b55de342d

Useful for figuring out how to use genism.

https://medium.com/@limavallantin/why-is-removing-stop-words-not-always-a-good-idea-c8d35bd77214

https://www.quora.com/How-can-word2vec-be-used-for-sentiment-analysis

This made me realise maybe a trained "Word2Vec" on a large corpora is better than what I am currently doing building a corpus out of my 100 articles.

https://www.oreilly.com/content/how-do-i-compare-document-similarity-using-python/

https://dev.to/coderasha/compare-documents-similarity-using-python-nlp-4odp#comments

These are the first articles that contains genism.similarities.similarity

Documentation on:

Bag_of_Words: https://www.kite.com/python/docs/gensim.corpora.Dictionary.doc2bow

Word2vec : https://radimrehurek.com/gensim/models/word2vec.html

Doc2vec: https://radimrehurek.com/gensim/models/doc2vec.html

Genism.similarities.Similarity: https://tedboy.github.io/nlps/generated/generated/gensim.similarities.Similarity.html

Gensim.corpora.Dictionary: https://radimrehurek.com/gensim/corpora/dictionary.html

https://www.tutorialspoint.com/gensim/gensim_transformations.htm

This website not only taught me that Tf-Idf may be inappropriate as I think we want the increased words ("Malware") to be significant in comparison.

https://dev.to/coderasha/compare-documents-similarity-using-python-nlp-4odp

This was instrumental in helping me use genism to calculate values of similarities. It helped me understand that I maybe don't need to use Tf-Idf but will experiment and check results.

https://stackoverflow.com/questions/18429143/strip-punctuation-with-regex-python/50985687

Useful in removing punctuation

https://sunscrapers.com/blog/8-best-python-natural-language-processing-nlp-libraries/

This gave me the idea of using Gensim for semantic similarity.

https://stackabuse.com/implementing-word2vec-with-gensim-library-in-python/

for using Word2vec, could maybe find a nice way of visualising word2vec.wv.vocab

I thought I might have to look at corpus's for bigrams within the word2vec model. Looking at the following websites below:

https://towardsdatascience.com/word2vec-for-phrases-learning-embeddings-for-more-than-one-word-727b6cf723cf

https://kavita-ganesan.com/how-to-incorporate-phrases-into-word2vec-a-text-mining-approach/#.YIF9sehKhPY

However I felt these were too complex for what I was after. I wasn't interested in any other bigrams really apart from the words I want, and so with that although it will mean I lose some semantic meanings as all "phrases" won't be acknowledged, if I changed my algorithm in pre-processing by replacing "targeted threat" with "targetedthreat", "malicious bot" with "maliciousbot", etc. This would be an easy way around the issue without a big impact on loss of semantics more than my model was already at.