

# Data Collection and Cleaning Coursework

## Introduction

For the implementation of this coursework, I have implemented a main() function which calls separate functions to perform each of the tasks. The functions used for each task are separated. I have also combined the solution for Problem 1 & 2.

## Problem 1 & 2

The solution for these 2 problems is combined. As soon as an article is read from the BBC website, it gets scrapped using BeautifulSoup. I have done this because I felt it was unnecessary to store all the data of a whole webpage, when we only require a small portion of the HTML for the algorithm.

The content of 100 articles pertaining to a single keyword is stored in the folder, articles within a single text file, "{keyword}.txt". This file will contain the title and body of 100 articles. The data of each article is separated by a block of '#' characters. All uppercase letters are converted to lower case.

The maximum number of articles downloaded per keyword, can be edited within the main function. Refer to the variable 'num\_articles'.

## Problem 3

### Preface:

The algorithm I have written is based on GloVe(Global Vectors for word representation). This is a pre-trained machine learning model which assigns vector values for each word. Within the data folder, there is a file called 'glove.6B.50d.txt'. This file contains the GloVe model. Do not delete this file as it is required for the algorithm to function. This model is publicly available at <https://www.kaggle.com/watts2/glove6b50dtxt>.

The basic concept of the algorithm is to assign vector values to each relevant word found in the articles of each keyword. These values are then compared with the vector values of other words using cosine the cosine distance between them. Cosine distance is calculated by finding the angle between to vectors and finding applying the cosine function.  $1 - \text{cosine distance} = \text{cosine similarity}$ . If the angle between 2 vectors is 0 degrees, there is no difference between the 2 vectors. Essentially keywords are compared, using vector values assigned to each word obtained from the articles relating to the keyword. The processes of the algorithm will be explained in detail over the next few paragraphs. The concept behind the algorithm was inspired by this article: <https://medium.com/@adriensieg/text-similarities-da019229c894>

### **Data Cleaning:**

The raw data obtained through the implementation of Problem 2 needs to be filtered. This data contains numbers, special characters, Stopwords and invalid words. To remove all non-letter, characters, I have used the module regex to only allow letters and spaces. The extra spaces are trimmed using the split() function offered by python. The resulting list is called 'words\_only' as it contains every word within the articles.

The next step is to remove Stopwords. Stopwords are extremely common words (the, he etc.), which are ignored by search engines. They tend to pollute the data because they appear frequently. A list of Stopwords can be obtained from the nltk module. These words are then removed from the raw data and a list called 'cleaned\_words' is obtained. Words which are 2 letters or smaller are also filtered out. Same goes for any word which cannot be found on the GloVe model. All data cleaning processes are done within the 'format\_text()' function.

### **Data Transformation/Formatting:**

This section involves the conversion of a list('cleaned\_words') to a dictionary with 2 traits. Format of the dictionary:

*Word: {frequency = int, vector = numpy\_array}*

As you can see the keys of the dictionary contain each word of the articles. The 2 traits given to each word are 'frequency' and 'vector'.

'frequency' contains the number of occurrences of a word within the 100 articles.

'vector' contains the vector value of a word as per the GloVe module.

This step is done within the 'format\_text()' function. A dictionary is obtained for each keyword.

The dictionary obtained will have a length of around 6000-7500 for 100 articles. This is a huge amount and is too large to be input to the algorithm. Comparing 2 dictionaries of lengths 5000 would take around 25 minutes. It is not viable to feed these large dictionaries to the algorithm. To overcome this issue, I have shortened each dictionary to contain up to 250 words with the highest frequencies. Total runtime for this solution(for 10 keywords) is not more than 5 minutes. This also helps in getting rid of some irrelevant words which only occur once and have little to no relation to the keyword.

### **Initial rejected Algorithm:**

My initial solution to this problem was to calculate the mean value of each vector of each word for all keywords and then compare them. This is a very naïve solution, especially for large amounts of data. This algorithm found all keywords to be extremely similar. The lowest similarity(%) value found using this algorithm was 97.2% and the highest was 99.5%. Just by looking at these values, you can tell something is wrong.

## Final Algorithm:

The algorithm takes 2 dictionaries as inputs. Each dictionary belongs to a specific keyword. These dictionaries contain the frequency and vector of each significant word pertaining to a keyword.

Each word within the first dictionary is compared with every word in the secondary dictionary. The score obtained for comparing the 2 words  $w_1$  from dictionary 1 and  $w_2$  from dictionary would be:

$$s_{12} = \sqrt{f_1 * f_2} * \sqrt{l_1 * l_2} * (1 - \cos(v_1, v_2))$$

$f_1, f_2$ : Frequencies;  $l_1, l_2$ : Word lengths ;  $v_1, v_2$ : Vectors;  $s_{12}$ : Score obtained

$$\text{Let } k_{12} = \sqrt{f_1 * f_2} * \sqrt{l_1 * l_2}$$

The total score obtained is obtained through the following equation:

**Total Score = Sum( $k_{ij} * (1 - \cos(v_i, v_j))$ )** where,

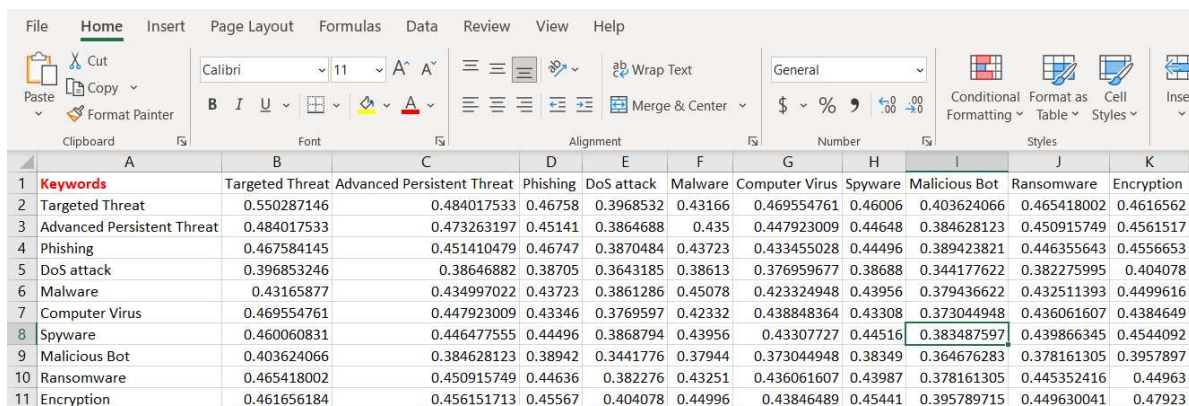
$i \in \text{every element in dictionary 1}, j \in \text{every element in dictionary 2}$

From this score we can calculate the average score from each comparison denoted by the equation:

$$\text{Average Score} = (\text{Total Score}) / (\text{Sum}(k_{ij}))$$

This average score is calculated for each combination of keywords and stored within a matrix. The average score between any 2 words is generally in the range of 0.3-0.6. Each dictionary is also compared with itself to provide a metric for the calculation of % similarity. These values are stored within the diagonals of the matrix.

The matrix will look something like this(this isn't the final result):



	A	B	C	D	E	F	G	H	I	J	K
1 Keywords	Targeted Threat	Advanced Persistent Threat	Phishing	DoS attack	Malware	Computer Virus	Spyware	Malicious Bot	Ransomware	Encryption	
2 Targeted Threat	0.550287146	0.484017533	0.46758	0.3968532	0.43166	0.469554761	0.46006	0.403624066	0.465418002	0.4616562	
3 Advanced Persistent Threat	0.484017533	0.473263197	0.45141	0.3864688	0.435	0.447923009	0.44648	0.384628123	0.450915749	0.4561517	
4 Phishing	0.467584145	0.451410479	0.46747	0.3870484	0.43723	0.433455028	0.44496	0.389423821	0.446355643	0.4556653	
5 DoS attack	0.396853246	0.38646882	0.38705	0.3643185	0.38613	0.376959677	0.38688	0.344177622	0.382275995	0.404078	
6 Malware	0.43165877	0.434997022	0.43723	0.3861286	0.45078	0.423324948	0.43956	0.379436622	0.432511393	0.4499616	
7 Computer Virus	0.469554761	0.447923009	0.43346	0.3769597	0.42332	0.438848364	0.43308	0.373044948	0.436061607	0.4384649	
8 Spyware	0.460060831	0.446477555	0.44496	0.3868794	0.43956	0.43307727	0.44516	0.383487597	0.439866345	0.4544092	
9 Malicious Bot	0.403624066	0.384628123	0.38942	0.3441776	0.37944	0.373044948	0.38349	0.364676283	0.378161305	0.3957897	
10 Ransomware	0.465418002	0.450915749	0.44636	0.382276	0.43251	0.436061607	0.43987	0.378161305	0.445352416	0.44963	
11 Encryption	0.461656184	0.456151713	0.45567	0.404078	0.44996	0.43846489	0.45441	0.395789715	0.449630041	0.47923	

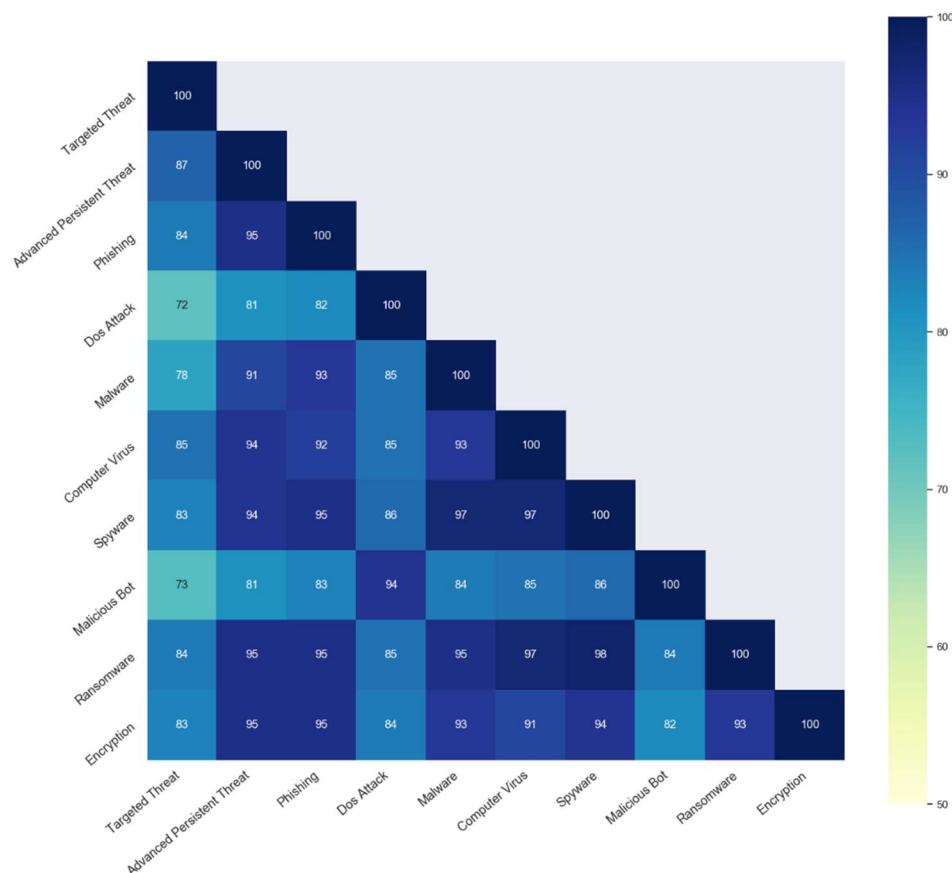
Each element in this matrix is then divided either by the diagonal element within it's row or by the diagonal element within it's column(The larger one out of the 2 is preferred). Each element is then multiplied by 100 to obtain the % semantic similarity.

## Problem 4

### Preface:

I have implemented 2 separate visualization techniques using seaborn to visualize the data obtained through the algorithm. The first is a heatmap which showcases all distances computed by the algorithm. The second is a bar-plot for each keyword, showcasing the semantic distance between that specific keyword and every other keyword. When the code for problem 4 is run ,the images generated are saved to the folder 'data' under 'images'. The images used here, along with some others are saved there.

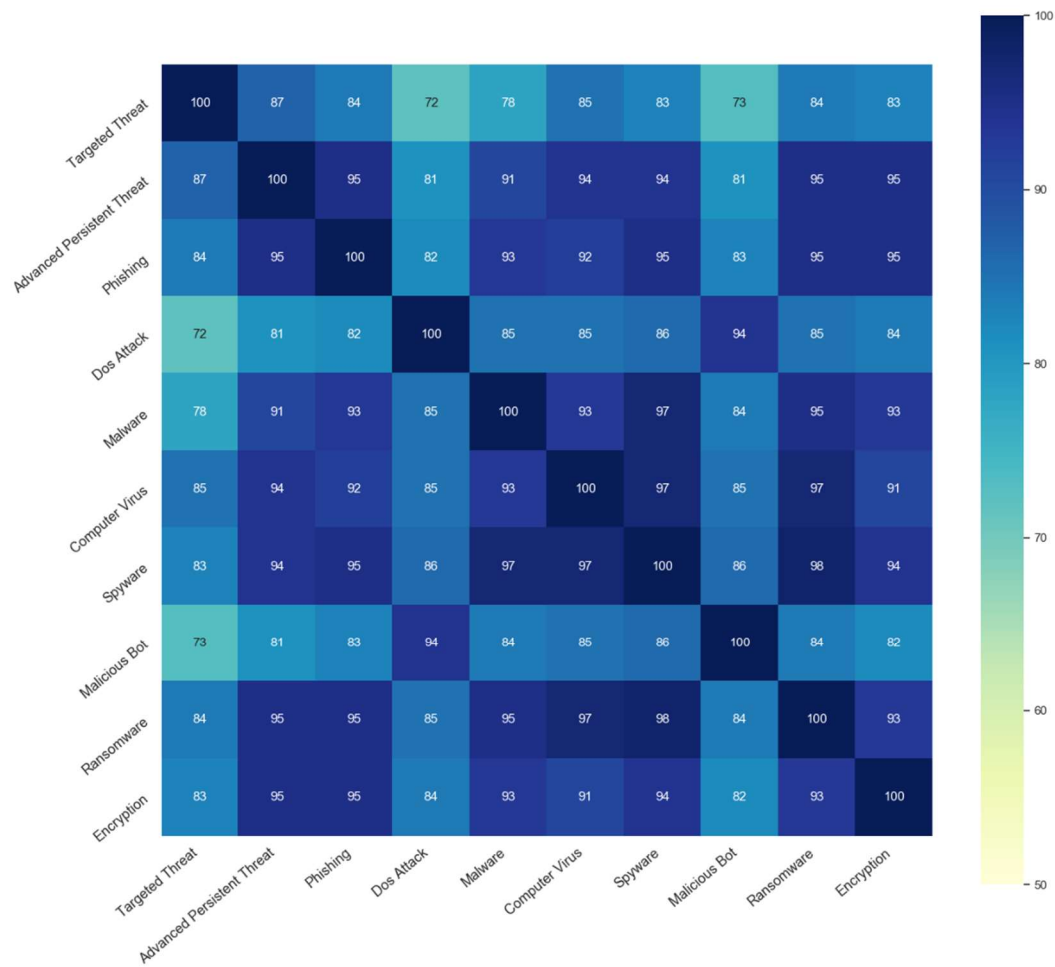
### Heatmap:



The input for this graph is a matrix of the % semantic similarity between each keyword.

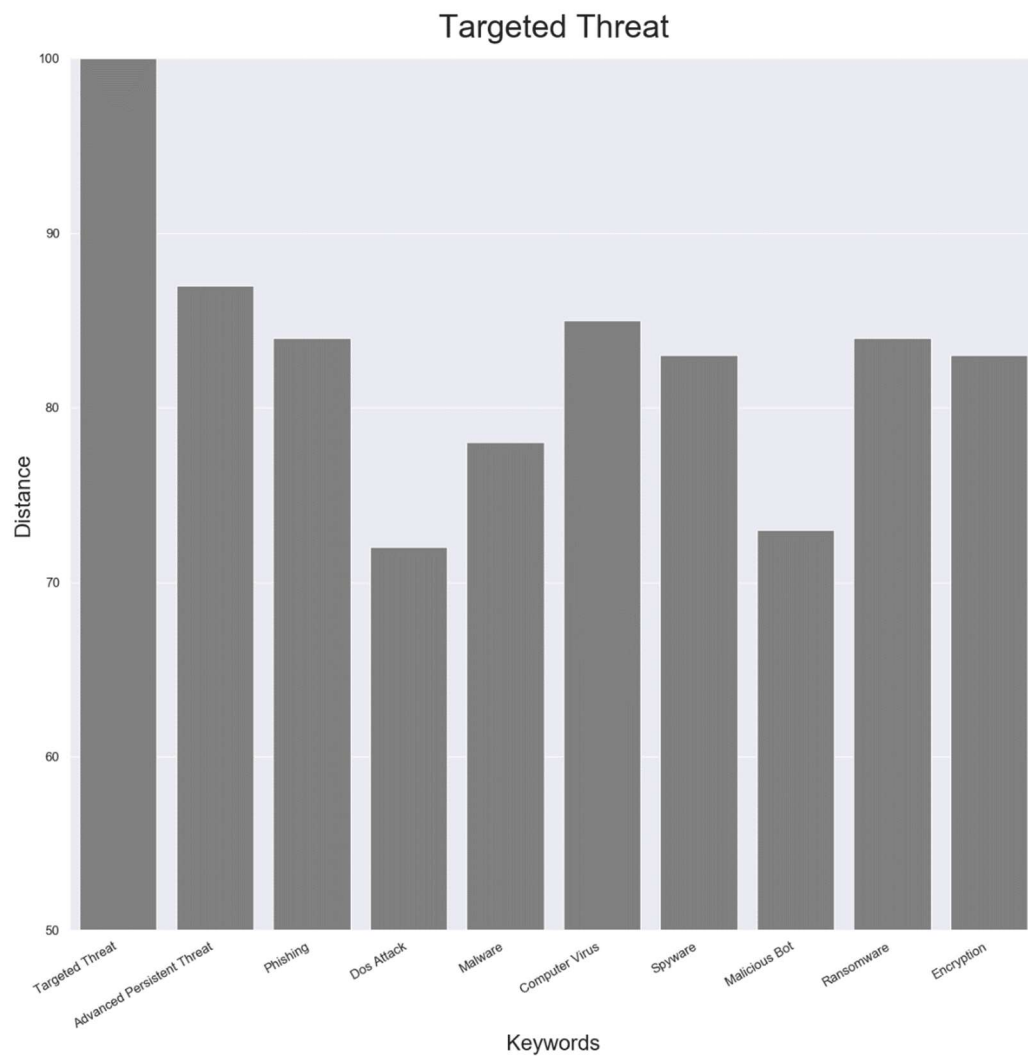
As you can see the keywords make up both the X axis and Y axis of this graph. A darker square means the 2 corresponding keywords are more similar. The number on each square is the % semantic similarity. The missing half of the heatmap is essentially a mirror of the bottom half.

Here is an image with the missing half:



### Bar-plot:

A bar-plot is generated for every keyword, it contains the % semantic similarity between each keyword it was compared with. This is useful if a user is more interested in viewing data of a singular keyword. The data presented by the heatmap can be overwhelming and a bit difficult to read especially for larger amounts of keywords. The bar-plot generated for the keyword 'Targeted Threat' is displayed on the next page. The bar-plot takes the keywords it was compared against as it's X axis and the % semantic similarity as it's Y axis.



As you can see, the data presented here is very concise and easy to take in.

The Heatmap is better if you want a summary, whereas the bar-plots are nicer if you want a closer look.