Detection of Suicidal Tendencies in Personal Reflections and Conversations: A Case Study of User Tweets

Damilare Samuel Adeola

A dissertation submitted to
The School of Computing Sciences of the University of East Anglia
in partial fulfilment of the requirements for the degree of
MASTER OF SCIENCE
AUGUST, 2022

SUPERVISOR(S), MARKERS/CHECKER AND ORGANISER

The undersigned hereby certify that the markers have independently marked the dissertation entitled **"Detection of Suicidal Tendencies in Personal Reflections and Conversations: A Case Study of User Tweets"** by **Damilare Samuel Adeola**, and the external examiner has checked the marking, in accordance with the marking criteria and the requirements for the degree of **Master of Science**.

Supervisor: _____

Dr. Tahmina Zebin

Markers: _____

Marker 1: Dr. Tahmina Zebin

_____

Marker 2: Prof. Shaun Parsley

External Examiner: _____

Checker/Moderator

Moderator: _____

Dr. Wenjia Wang

DISSERTATION INFORMATION AND STATEMENT

Dissertation Submission Date: **August, 2022**

Student:	**Damilare Samuel Adeola**
Title:	**Detection of Suicidal Tendencies in Personal Reflections and Conversations: A Case Study of User Tweets**
School:	**Computing Sciences**
Course:	**Computing Science**
Degree:	**MSc.**
Duration:	**2021-2022**
Organiser:	**Dr. Wenjia Wang**

STATEMENT

Unless otherwise noted or referenced in the text, the work described in this dissertation is, to the best of my knowledge and belief, my own work. It has not been submitted, either in whole or in part for any degree at this or any other academic or professional institution.

Subject to the confidentiality restriction if stated,m permission is herewith granted to the University of East Anglia to circulate and to have copied for non-commercial purposes, at its discretion, the above title upon the request of individuals or institutions.

Signature of Student

Abstract

Suicidal Detection and treatment from the clinical and public health perspective is reactive. For an action whose consequences are irreversible, a reactive approach to the problem cannot be the answer. A proactive approach is needed to solve and detect Suicidal Intent. Social Media has become the television and diary of millennials and gen z alike, hence it's imperative to create techniques and approaches to study their actions in this particular space. This research involved creating Document Similarity Algorithms from Corpora mined from the Twitter Developer API. Making the data unique to this platform; a methodology design involving validating data at various spectrum and selecting an appropriate threshold to classify the similarity levels were created as well as a lexicon unique to the Twitter Dataset. With an accuracy score of 84%, the Jaccard Document Similarity Algorithm was able to spot Suicidal intent from User's Tweets and with an accuracy of 93% was also able to spot non-suicidal intent. The Jaccard model seemed to be the most durable and effective for the problem and was chosen as the algorithm for detecting Suicidal Tendencies in Users' Tweets.

Acknowledgements

I dedicate this research and project first of all to Olodumare, for giving me the grace and fortitude to finish my Master's program well. I also would love to thank my mother, Gloria Sholape Ogunjobi, for her undying support, care and love, without which I am a leaf in the wind. A lot of thanks also goes to Chika'm, the love of my life, for pestering me to finish on time and ever present when I was down. A lot of thanks goes to Dr. Wenjia, for his endless understanding, support and attention. Also, I would like to say a big thank you to my supervisor, Dr. Tahmina, for her grace for taking me under her wings despite the uncommon circumstances. Lastly, I want to thank my friends, in no particular order, Rufus, Marvel, Maz, Raj, Ally, Ben, Esther and the many more that have skipped my mind, thank you.

Damilare Samuel Adeola

Norwich, UK.

Table of Contents

List of Figures

1 Introduction

It has been estimated that 4,912 people took their own life in 2020 in England(samaritan.org 2020). That means on a daily average, 13 people took their own life each day in 2020. The suicide rate was 10 per 100,000 people(samaritan.org 2020), and in figure 1, you would see that the amount of people committing suicide has been steadily increasing over the years.

Since the rate of suicide has been steadily increasing over the years, what have Primary Care Providers(PCP) been doing to mitigate this? What have the Clinical Psychiatrist and Psychologists been doing to plug this hole? The answer is not simple. Several researches have emerged whereby they sought to use Natural Language Processing(NLP) to create a model which can detect the likelihood of someone committing suicide from their text conversations at a fairly accurate rate and more importantly timely manner.

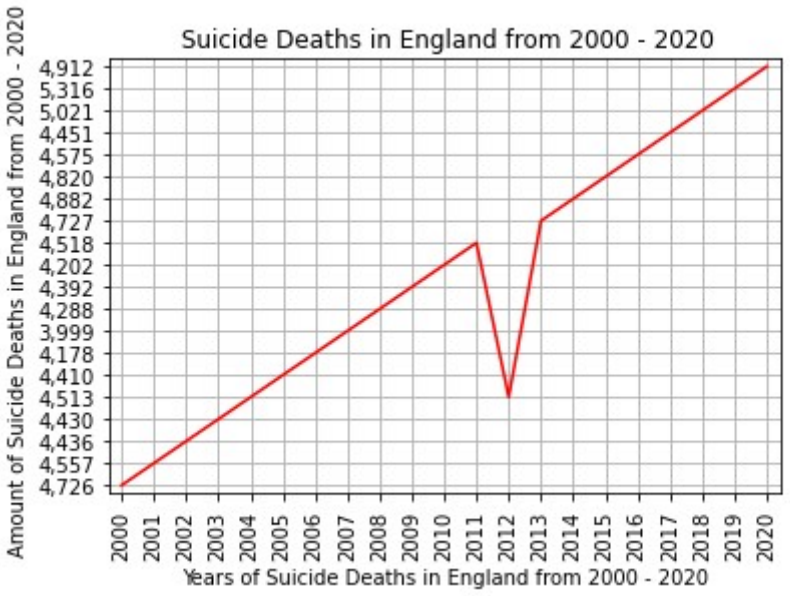

Figure 1: A plot showing the amount of people committing suicide annually in England from 2000 - 2020

Primary Care Providers, Psychiatrist and Psychologists actually do detect suicidal tendencies but they only detect these tendencies when patients with such suicidal tendencies come to them for personal consultation(Coppersmith et al. 2018). Hence, in the Health Care sector, there is a reactive solution to suicide prevention rather than a pro-active solution. And this is no fault of theirs as a PCP can only diagnose and treat what they are aware of. The use of Natural Language Processing(NLP) to be proactive in combating suicide prevention would give PCP and other Health Practitioners in the Mental Health Sector, the necessary tools to be proactive in not only treating people who have come to them with suicidal tendencies but to also identify people who have or are about to have suicidal tendencies.

The power to be able to predict via Social Media conversations and reflections, if a particular user has suicidal tendencies is what makes this approach effective. By using NLP and Machine Learning to perform Sentiment Analysis with Twitter data, an algorithm to detect suicidal tendencies in user tweets can be created so as to enable the relevant health stakeholders to help

such user and further save a life.

1.1 Background to the study

Suicide is one of those topics that is considered a taboo in lots of African Culture. This is because some African Cultures do not believe that humans own their life, they believe that this life was given to us by a supernatural being. In some practises, like the Yoruba's of Nigeria, the Oba (King) would be given either an empty calabash or a parrot's egg or skull, called Aroko, a form of long-distance communication, to communicate to the king to commit the ultimate sacrifice by taking his life (Solanke et al. 2022). In this instance, suicide is a form of punishment but in reality, Suicide is seen as a form of release, to escape pain, hurt, depression, a permanent coping mechanism or as self-blame which was well expatiated in the work by Brevard et al. (1990).

There's also a personal nature for me to this topic; as an individual who sometimes goes through bouts of depression and melancholy, the thoughts of suicide have indeed festered from time to time. However, I have found that if one focuses on love and not the pain or sadness, it is a relief, the nature of this relief is undecided as at now, but I feel the relief whenever I talk to a loved one or I remember a loved one. Perhaps there may be a correlation between suicide and loneliness?

Social Media has become a force in our lives, and with it it's attendant effects like increase in narcississm, self-absorption, selfishness, envy etc. There are studies that have looked at how Social Media has increased the rates of suicide like that done by Luxton et al. (2012) whereby they looked at the effect of how the internet and social media can influence suicide-related behaviour. It is for this reason that it is important for there to be research or rather increased research that can predict with a good level of accuracy if a particular user, through their interaction on their Social Media of choice, has suicidal tendencies or not.

You must note that in the last sentence of the previous paragraph, no attempt was made as to the type of media. This is deliberate, because even though this research focuses on the textual aspect of Twitter as this is what the Social Media platform was built for, to amplify one's thoughts, the approaches and algorithms elaborated in Chapters 3 and 4 can be extended to other Social Media platforms that use other forms of media like pictures for Instagram or short videos for Tiktok - as the paramount outcome is to be able to detect Suicidal Tendencies whenever user's are on Social Media.

1.2 Relevance of the Study

Digital Phenotyping can be defined as capturing and measuring the real-time features and quantification of human behaviour(Bidargaddi et al. 2017). With the amount of screen time spent by humans increasing every year, Fig 2 shows a horizontal bar chart of total screen time spent by people in different countries. The red bar is that of United Kingdom. It is estimated that in the UK, 50% of adults look at a computer screen for 4 hours or more each day and 26% of adults look at a smartphone screen for 6hours or more each day(Clayton & Clayton 2022). This means that 50% of adults spend 16.67% of their time each day looking at their smartphone screen. This statistic is given more context if we subtract the average sleep time

from the 24hours so we can calculate the percentage of screen time as a proportion of when adults are awake. It is estimated that the average Briton gets just 6hours 19minutes sleep per night(Hall 2018). This means the average Briton is awake for 17hours, 18minutes per day. Now to get the proportion of screen time from the average total awake time:

Total Awake Time = 17.81

Screen Time = 4

Proportion of screen time in total awake time = (4/17.81)*100 = 22.41%

This means that the average Briton spends 22.41% of their total awake time looking at the screen.

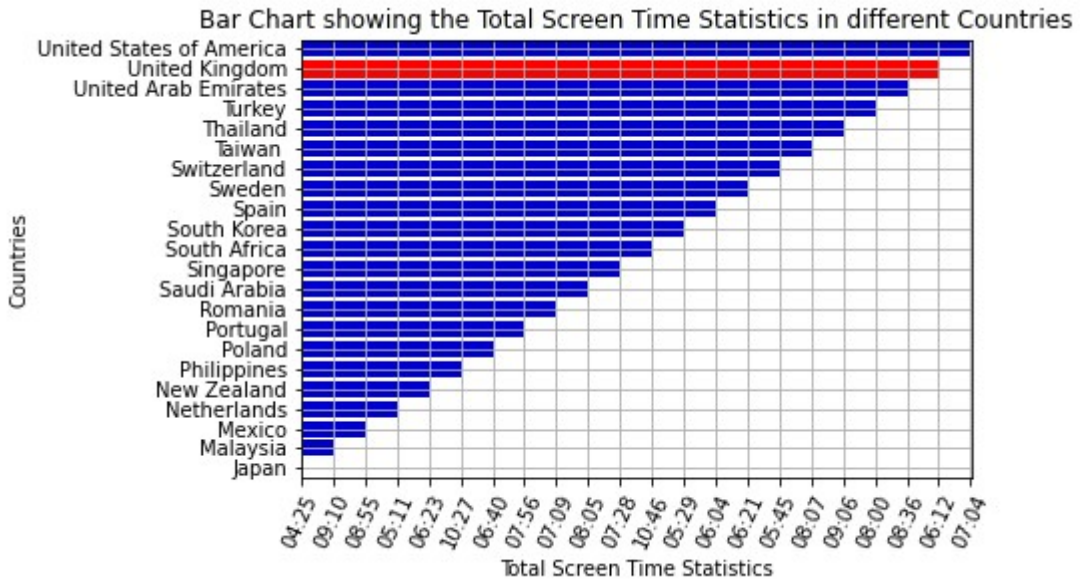

Figure 2: A bar chart of Total Screen Time by countries

What does this elaborate calculation mean to this study? It has great significance, what this means is that a lot of Digital Phenotyping is likely to happen among Britons, hence, we can actually use this data – digital conversations, reflections, tweets to be able to find out the suicidal tendency in a user's tweets since they spend a lot of time interacting in the digital world. Figure 3 shows a summary of the amount of screen time spent on social media.

1.3 Aims and Objectives

1. Build a Natural Language Model that would be able to predict the suicidal tendencies in a user's tweets.

2. Build a corpus of suicide text samples (this is not readily and easily available online).

3. Going further from the previous point, by building a corpora suicide text samples with categories and numerical scores for particular features.

4. Build a Natural Language Processing pipeline (Web Application) whereby a series of text can be passed through, and the NLP model would give a probabilistic estimate of whether that text contains suicidal thoughts.

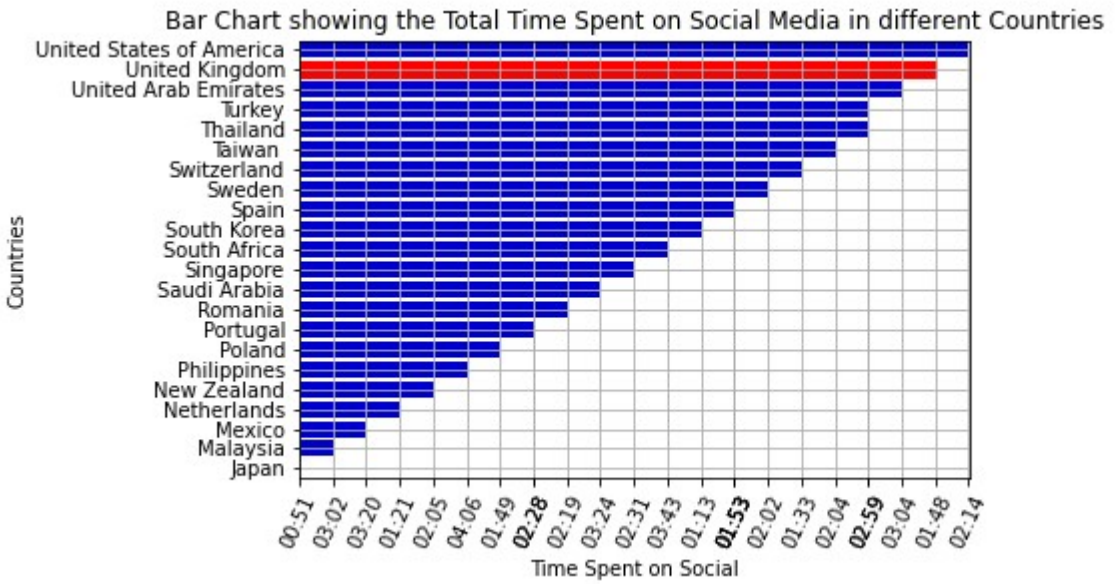

Figure 3: A bar chart of Total Social Media Screen Time by countries

1.4 Achieved Aims and Objectives

All the aims and objectives set out in the project were achieved as well as other goals which were not earlier stated but were also achieved in this research. They include:

1. Four Document Similarity Models were built and the adopted one for this research had an Accuracy rate of 84%.
2. A SWEETS(suicidal Window, Expressions, and Emotions in Tweets) CORPORA was built containing over 1,073,595 tokens.
3. A SWEETS CORPUS was built containing over 402,777 tokens.
4. A SWEETS MINI was built to be able to help when deploying the models, containing over 235,392 tokens.
5. A Suicidal Sentiment Lexicon(SSL) was built based off the data mined from the Twitter API.
6. A Twitter API cleaning Algorithm was built to enable faster pre-processing so researchers can focus on analysing and extracting necessary insight from textual data.
7. A suite of micro-services, all built from the data gotten for the Twitter API and deployed live to Heroku as containerized applications were built.

2 Literature Review

2.1 Deep Learning for Suicide and Depression Identification with Unsupervised Label Correction by Ayaan Haque

One of the major problems of Suicide and other mental illness NLP activities have is the dirge of data to be able to perform efficient and effective classification problems with Natural Language Processing. There is also the concern of noise in data. These two problems were the focus of the paper by Haque et al. (2021) whereby a novel clustering algorithm was developed to be able to efficient and effectively cluster noisy data thereby offering improved accuracy scores across a varied range of both baseline machine learning algorithms(knn, SVM) and Deep Learning Neural Networks(DNN). Their approach with the DNN offered high levels of accuracy after their algorithm had been able to de-noise the data.

The obvious reasons for removing noise in any Natural Language Processing task is to make sure that the algorithm is correctly trained and able to efficiently classify the data. They observed that in order to do this process, there were three main approaches that could be followed:

1. noise-robust methods (Haque et al. 2021)
2. noise-tolerant methods (Haque et al. 2021)
3. data cleaning methods (Haque et al. 2021)

The approach chosen for their research was the data cleaning method. The reasons for this were presented clearly, in the noise-robust method, it was realized that they are over-reliant on algorithms that tend to be naturally less sensitive to noise which means data that have low-dimensionality or low amount of features. And since we are aware that for most Natural Language Processing tasks, there is a stage called Vectorization or Word Embeddings whereby each word is mapped to a particular number(vector). This is how NLP Researchers are able to perform Natural Language tasks as the Computer cannot understand human-readable langfuage only, each word has to be transformed to a vector. This ultimately means that at the early process of any NLP task, there would be high-dimensonality of data or a high amount of features. This makes the noise-robust approaches almost impracticable in NLP tasks.

The authors presented a novel technique and algorithm for classifying depression and other suicidal tendencies using data that has been web-scraped and neural networks (Haque et al. 2021). The called their technique SDCNL which stands for insert it here. The process for the SDCNL technique is shown in the flow chart below:

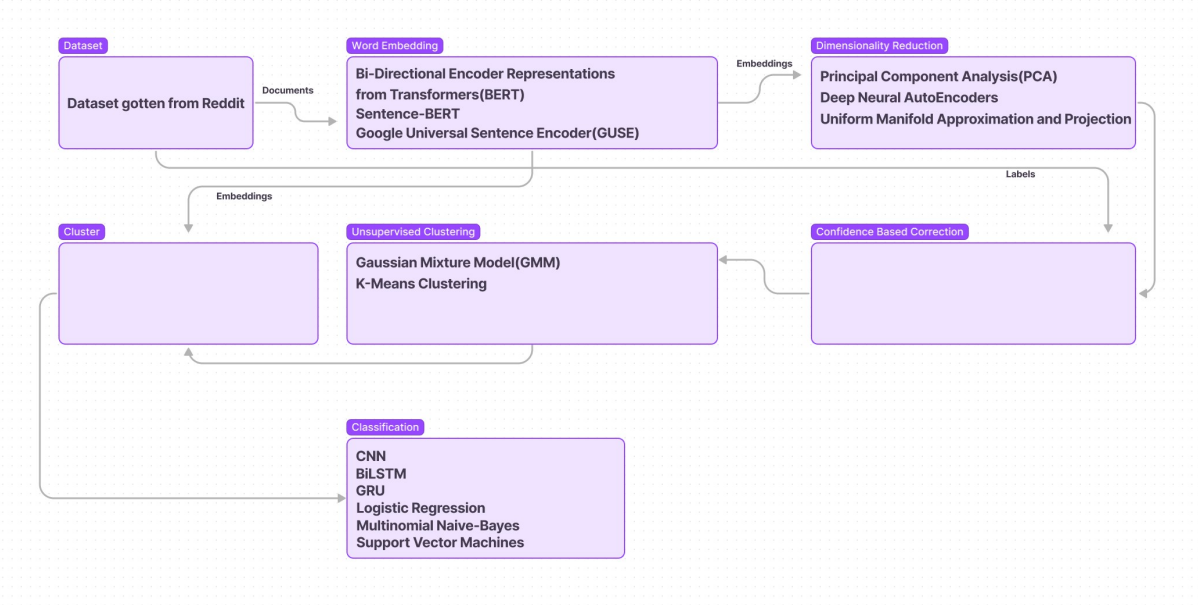

Figure 4: A process flow chart showing the various steps involved in the SDCNL Algorithm.

Figure 1 shows the process flow chart as described by (Haque et al. 2021) for their SDCNL algorithm. The process begins with collecting the dataset from Reddit, other datasets were used but this formed the bulk of their data and one of the main reasons for choosing this dataset was because of the anonymous nature/functionality present in using Reddit. The authors surmised that, since both depression and sucide are very sensitive topics, and in order to filter out false positive signals, they believed that individuals would be more open and honest about their experiences especially with sentitive and stigma prone topics like depression and suicide. I do agree with this premise for data collection and the results would be briefly explained lated in the following paragraphs. After the data collection stage, is the Word Embeddings/Vectorization stage. Here the authors made use of three major techniques for Word Embeddings and another three for the baselines. The majors were:

- Bi-directional Encoder Representations from Transformers(BERT)
- Sentence-BERT
- Google Universal Sentence Encoder(GUSE)

While the other base line Word Embeddings techniques used include:

- Term Frequency-Inverse Document Frequency(TF-IDF)
- Count Vectorizer(CVec)
- Hashing Vectorizer(HVec)

After converting each word in the document to a vector with each of the different Word Embedding Algorithms listed above, they proceeded to reduce the dimensionality of the data. In this stage, three different dimensionality reduction techniques were used:

1. Principal Component Analysis
2. Deep Neural AutoEncoders
3. Uniform Manifold Approximation and Projection(UMAP)

The next stage was the Confidence Based Correction. Here, the main task was to eliminate as much as possible the label noise and this is where they proposed an unsupervised label correction mechanism. After the dimensionality reduction, they used an unsupervised clustering algorithm to separate them into two distinct clusters (Haque et al. 2021). It was in this stage that GMM was used. This clusters were made up of the original labels gotten from Reddit which were the ground truth and the new labels arising from the unsupervised clustering (Haque et al. 2021). Then if the clustering algorithm classifies a label with a probability above a certain threshold, in the paper regarded as t, a tuned threshold, the ground truth is assumed (Haque et al. 2021).

After the label correction, deep neural networks were used to train the data. Two labels were created - depressive and suicidal sentiment. In this classification stage, any classification algorithms could be employed, but in thier research they made use four deep learning algorithms and three baselines.

The datasets used for the train and test data were gotten from two reddit sub threads - r/SuicideWatch and r/Depression. The dataset contained 1,895 total posts. And this was used to train the different algorithms during their experiemnts. The posts from the subreddit - r/SuicideWatch were labelled as suicide while the posts from the subreddit - r/Depression were labelled as depressed.

Furthermore, two more datasets were used in various capacity for the experiments. There were the Reddit Suicide C-SSRS dataset and the IMDB movie datatset. The former datatset contains 500 reddit posts from the subreddit r/Depression. One important thing to note about the Reddit Suicide C-SSRS datatset is that the posts were labeled byu psychologists as according to the Columbia Suicide Severity Rating Scale(C-SSRS), they assign progressive labels according to how severe the depression sentiment contained in the post is (Haque et al. 2021).

In the implementation stage, 20% of the data was used as the test/validation data. The tools used in the research included TensorFlow - a deep learning framework created by Google and Scikit-Learn - a popular Machine Learning Library. The deep learning algorithms were trained using the Adam Optimizer and used a Binary Cross-Entropy Loss function. In order to measure the performance of their model, the various classification metrics were used:

1. Accuracy Score(Acc)
2. Precision(Prec)
3. Recall(Rec)
4. F1-Score(F1)
5. Area Under Curve Score(AUC)

In order to test the clustering performance of their model, two accuracies were presented in the results - one is the classification performance of the clustering algorithm correcting the noisy

labels and the other is the classification performance after the label correction. The expectation is that, the accuracy performance and other classification metrics of the model would be greatly improved after the label correction as compared to before the labvel correction has beeen done(i.e the noisy labels).

Different noise levels were injected into the dataset. 10 - 40% of the dataset were corrupted with noise. This was done on purpose, in order to ascertain the perfomance strength of their Clustering Algorithm as efficiently separating label noise from the data. Their assumptuion was that close to 8 - 38% of labels in real-world datasets were beset with noise (Haque et al. 2021). Afer this, the performance of the Clustering Algorithm in correecting the noisy labels was evaluated (Haque et al. 2021). The results were good as the SDCNL label correction proved successful on both the IMDB dataset and the C-SSRS dataset, showing its utility and versatility in real-world datasets.

In the results, the best classification performance arose from the comboinations of BERT with CNN - BERT as the Word Embedding Algorithm while CNN as the deep learning algorithm. Figure 2 shows in various charts the performance of the different approaches of the SDCNL algporithm experiments done by the authors.

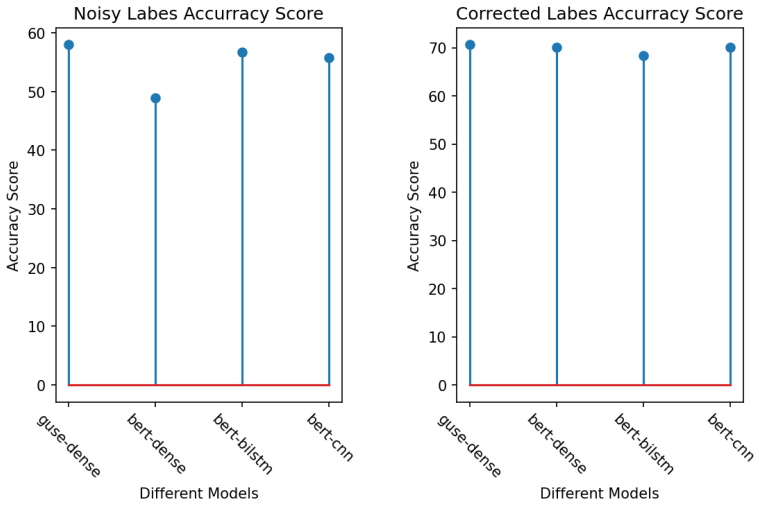

Figure 5: Stem plots showing the accuracy scores of both the noisy label and the corrected label.

As can be seen in Figure 2 that there is a difference in the accuracy performance of the model once label correction has taken place as opposed to when it has not - noisy label. This shows that the premise made for this stage of the research holds true and this would indeed increase the classification performance of the model as can be seen in the stem plots of figure 2.

Of all the various experiments performed, figure 3 illustrates the four best combinations of Word Embeddings and Classifiers according to various Classification metrics - Accuracy Score, Recall, Precision, F1 and AUC.

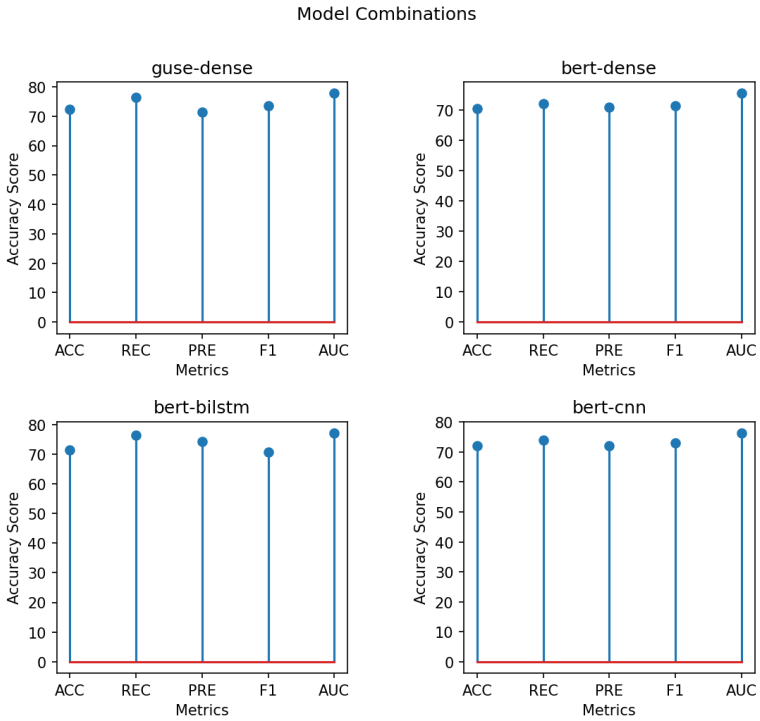

Figure 6: Classification Metrics performance of the top four combinations of Word Embeddings and Classifiers used in the research.

In figure 3, we can see that the combination of GUSE with a fully-dense neural network(guse-dense)guse-dense outperformed othermodel combinations in all the classification metrics except that of Precision. Because of the good performance of the above models combinations with neural networks - bert-cnn, bert-dense etc, the four model combinations were employed for the further experiments in the project, chosen over the baselines.

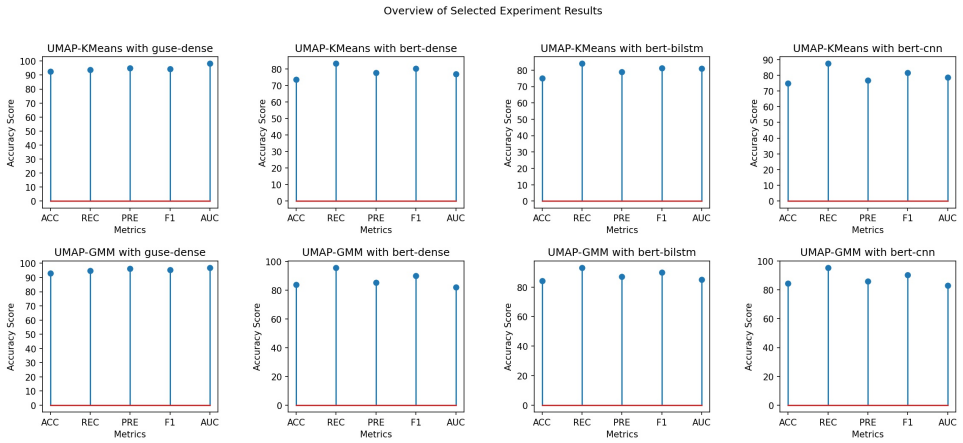

Figure 7: Classification Performance of the final selection of Word Embeddings, Dimensionality Reduction and Classifier combination.

In the final performance of the models with threshold label correction ad without, the figure 4, shows the various performance of the two major algorithms used for the classification in tandem with the Word Embeddings and Classifiers. These were the:

1. UMAP-KMeans
2. UMAP-GMM

The results concluded that their method greatly improved the models performance. Their label correction method, for example, improved the ROC curve and yielded a much higher AUC value. Which means that the model performed well in areas of Specificity and Sensitivty and hence can be concluded that the model is relevant. After all the experiments, the final model chosen involved a combination of GUSE as the embedding model, a fully dense neural network as the classifier, UMAP for the dimensionality reduction and a GMM for the clustering algorithm. Figure 5 below provides a visual summary of their proposed algorithm and clustering technique.

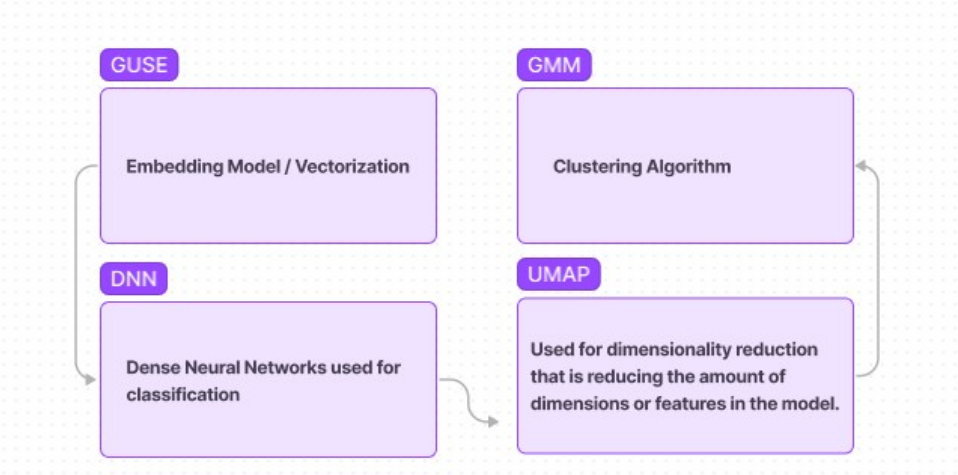

Figure 8: Visual Summary of the algorithm used in the research after all experiments have been performed. These were the most optimal combinations that yielded the best classification performance metric scores as shown in figure 4 earlier (Haque et al. 2021).

The best combination of Word Embeddings, Clustering, Dimensionality Reduction and Classifier combination involved using the guse-dense with UMAP-GMM. This combination prior to which label correction would have been done to remove the noise yielded very high classification performance results with an accuracy score of 93.08, Recall score of 96.16. The other classification metrics can be seen in figure 4 above. Overall, this combination proved to be the most powerful and successful of all the 42 experiments conducted in this research. The premise for this research was sound, finding a clustering algorithm to be able to remove noise - irrelevant data from a suicidal/depression themed word document using SDCNL. The technique was also tested in the wild with the IMDB data-set as well as the C-SSRS data-set and also proved to be as effective even in other NLP tasks that d not involve suicide or depression. The analysis and experiments were extensive - a total of 42 experiments were carried out. Their choice of data was good as I think the anonymity of the Reddit social media platform does provide truly, a honest feedback, and consequentially a safe place whereby people who are suicidal can express themselves without fear of judgement. The only concern here was the amount of data used to train the neural network. From their Reddit web crawler code, they were able to collect a total of 1,5067 posts. For an NLP tasks, this is inadequate. However, despite this shortcoming, its impact was absent in the model's performance as it performed well in both the training/validation

data as well as data that was even thematically different from what the neural network was trained on. This gives credibility to the authors clustering technique.

A good amount of works have been published relating to using Natural Language Processing to detect and then classify users suicidal tendencies in Social Media. Apart from the increasing amount of screen time especially among teenagers to young adults (Boers et al. 2019), the deluge of information and live data has drawn researchers to using Social Media data - and this ranges from Facebook to Reddit to Twitter.

Twitter in particular has proven to be quite important in the study of mental health related issues of recent. Apart from the Samaritan 'Radar' scandal in 2014, there have been quite positive outcomes from using the Social Media's data in creating models for detecting suicidal tendencies. Twitter's text format and limited amount of characters make it a platform best suited for expressing one's thoughts and in use, that's how it feels, this is why it took a long time for Twitter to introduce the ability to delete tweets because really can we delete our thoughts?

These thoughts can also be suicidal, and Twitter has recognised it's role and put adequate services in place to help people (Coppersmith et al. 2018). But these services are not in real time. Research has been done whereby the data gotten from Twitter were from two groups: users who have tried committing suicide in the past and those who have not attempted to commit suicide (Coppersmith et al. 2018). This was done so the algorithm can be trained to be able to differentiate between those who are at risk of committing suicide and hence in urgent need of PCP and those who are not. Samples were also gotten of those who were a close match to having this suicidal thoughts but have not yet attempted suicide.

Two data sets were used in the experiments, one gotten from Twitter and the other from the now defunct ourdatahelps.org. When the two data sources were combined, they was a total of 547 users who have tried committing suicide and 418 users who actually committed suicide(Coppersmith et al. 2018). The categories of their data were fairly mixed, with most of the population comprising of females aged 18 to 24.

Despite the limited sample size, Deep Learning was used in trying to categorize the data. However, various layers were added to the layer to compensate for this shortcoming, like the use of the GloVe Embeddings(Glove stands for Global Vectors and it has to do with mapping the vectors of words to a global space(Pennington et al. 2014)). These various layers were introduced in order to prevent the model from overfitting with the training data. These word embeddings were later processed to make them better at capturing language relating to mental health(Coppersmith et al. 2018). The last layer involved using a 'sequences of word vectors which are processed via a bi-directional Long Short-Term Memory(LSTM) layer to capture contextual information between words'(Coppersmith et al. 2018).

There were two key results from their research, one was that the machine learning algorithms with high precision could delineate through text from social media those who would go on to attempt suicide and those who would not and secondly, that machine learning algorithms rely on a large amount of little 'clues' rather than whole phrases(Coppersmith et al. 2018).

Deep Learning always gives amazing results when it comes to classification problems, however, it comes at a high computational cost. Also, it stands to reason if for such a small sample size, if it was worth this cost. The results though tell otherwise, with their model having a true

positive rate of 84% (Coppersmith et al. 2018).

2.2 ScAN: Suicide Attempt and Ideation Events Dataset by Bhanu et al

In this paper, the authors research death on creating a dataset into two sub-categories:

1. Suicide Attempt(SA) : This was a tricky label in the research. It primarily had to deal with the instances in the EHR whereby the patient attempted to take his or her life. It is tricky in the sense that, there were some instances whereby it was unclear if the life-ending attempt was as a result of suicide or perhaps other factors. An example shared in the research mentioned a case of a patient 'tried to hang himself' (Rawat et al. 2022), in this instance, this was labelled as positive but in another record whereby the patient tried to take an overdose of Tylenol, this was labelled as 'unsure' because the overdose was never confirmed as a suicide attempt in the EHR of the patient's hospital stay (Rawat et al. 2022).

2. Suicide Ideation(SI) : can be defined as any moment in EHR whereby the patient made a mention of wanting to take his/her life or cause his or herself any harm. During the annotation phase, an SI could be labelled either positive or negative.

They named this dataset ScAN and it is publicly available. This alone is welcome development in the Mental Health and Suicide NLP field whereby, sometimes getting data to build models can be excruciatingly difficult. In this paper, the authors not only developed a novel data based on Electronic Health Records(EHR) (Rawat et al. 2022), they also made the data publicly available.

The dataset consisits of 12,759 EHR note (Rawat et al. 2022). In order to be able to process these EHR's, a process of annotation had to take place whereby the annotation was done in the presence of a Mental Health expert – work on this. This process in the research was the Data Collection aspect and it had a few hiccups. The first of which was in the class labels. There were a total of three class labels making the problem a Multi-Class Classification problem. The class labels were Suicide Ideation(SI), Suicide Attempt(SA) AND Negative Unsure. The Negative Unsure Class label arose during the annotation phase. Here, as the annotations were taking place, there were some EHR's in which it was unsure if the patient actually had Suicide Ideations(SI); but upon consultations with the Health Practitioner present during the annotations, a new class label was created called Negative Unsure.

The second part of this research was called ScANER(Suicide Attempt and Ideation Events Retriever) (Rawat et al. 2022), this is a NLP model based on a form of Google's Bert Model called the RoBerta model. The RoBerta model is similar to the Bert model only that in this case, the Bert model has been further trained and optimized. Also, it was noted in the research how specifically trained or thematically trained NLP models perform better than base-line or broad based ones. For example in the Health domain, we have the clinicalBert (Alsentzer et al. 2019), BioBert (Lee et al. 2020) all extensions of the Bert model. The authors also presented their own model and named it: medRoBerta. It is a pre-trained RoBerta base model that uses the MIMIC(expatiate) dataset to create a clinical version of the RoBerta model.

The figure 6 below summarizes the algorithm for the research used in building ScAN.

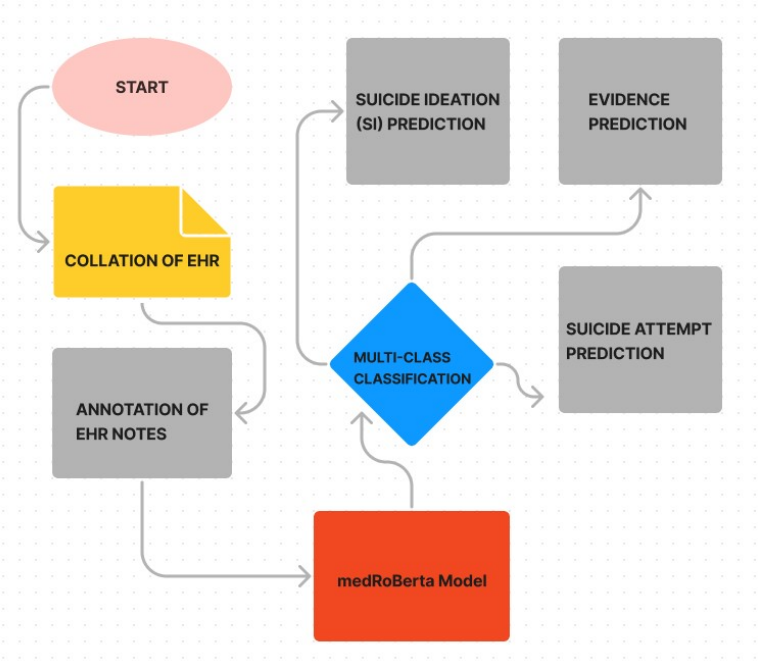

Figure 9: A flow chart showing the algorithm for ScAN

The medRoBerta was trained in a multi-task learned environment. It was trained on two fronts:

1. ability to identify the label for Suicide Attempt between the various intra-classes - positive, negative, unsure and neutral-SA
2. ability to identify the label for Suicide Ideation between the various intra-class labels - positive, negative and neutral-SI

The figure below shows a figure that summarizes the statistics of the various intra-class sub-labels in both Suicide Attempt and Suicide Ideation. It provides a visual snapshot of the spread of the data used in the research.

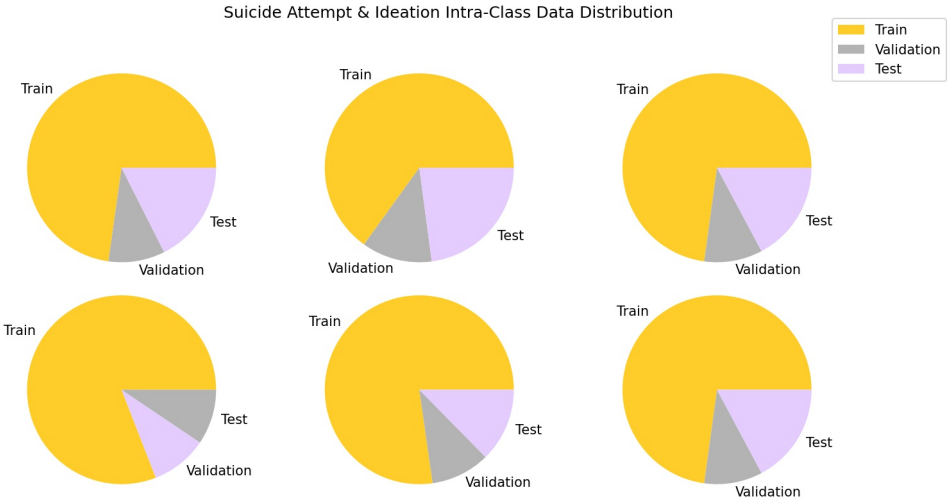

Figure 10: Figure showing the various intra-class distributions between train, test and validation data for the(from top-left to lower-right) - SA Positive, SA Negative Unsure, SA Neutral, SI Positive, SI Negative, SI Neutral

For the results of the research, three classification metrics were used to test the relevance of the medRoBerta model. There were the Precision, Recall and F-1 Score. While calculating the F-1 score, the maximum average was used, this was because this was a multi-class classification problem. The figure below provides a visual summary of the results.

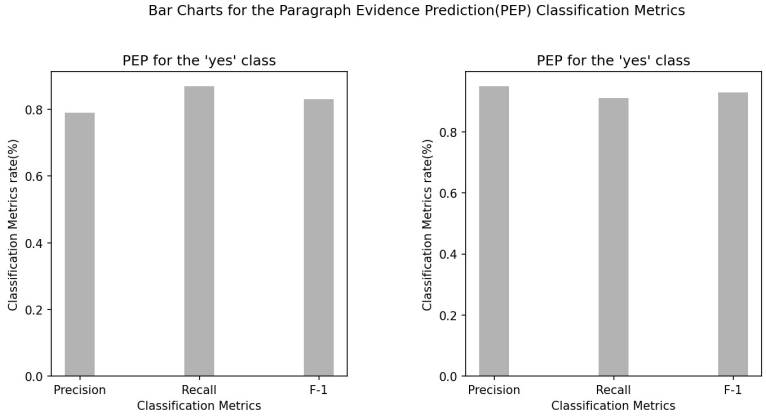

Figure 11: Caption

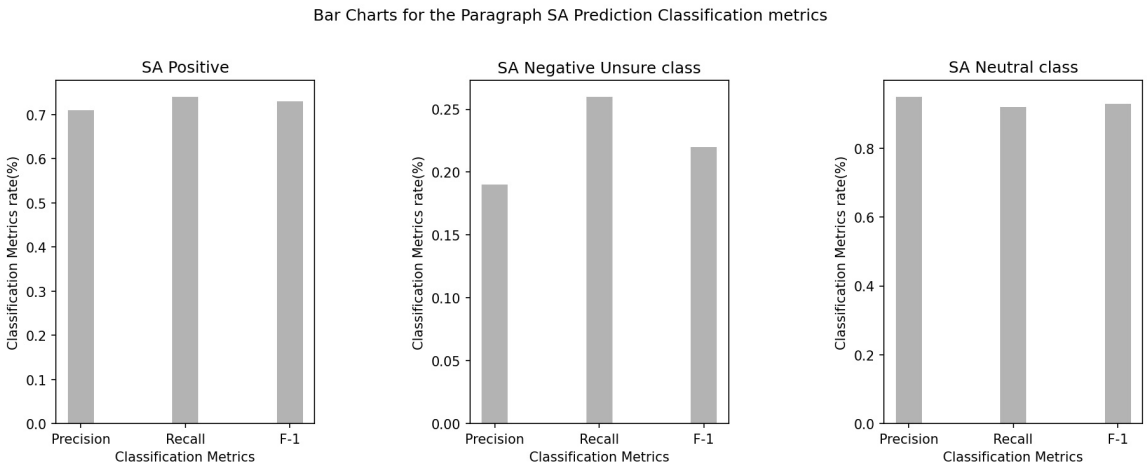

Figure 12: Caption

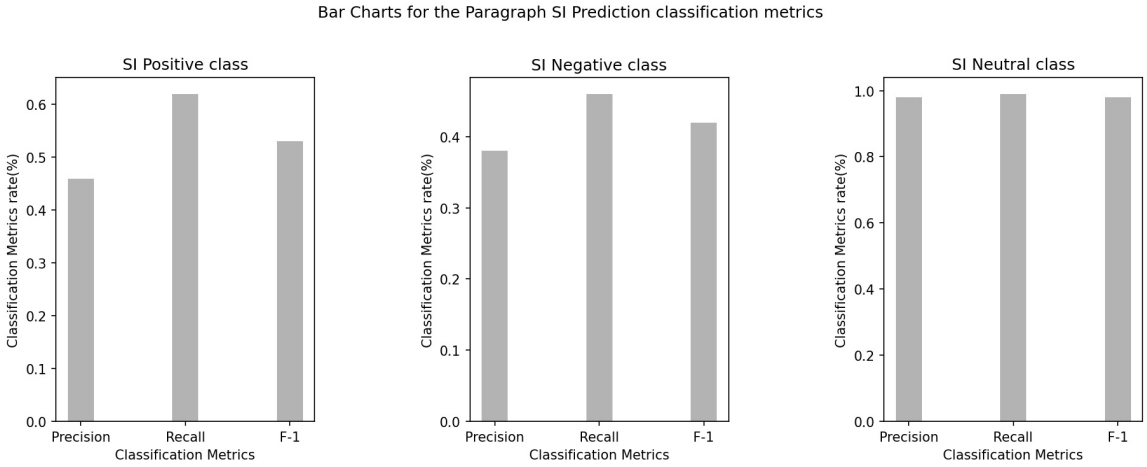

Figure 13: Caption

Figures 8, 9, 10 show the various classification metrics used in the research and their performance. The Paragraph Evidence Prediction(PEP) had good classification performance in both the Yes and No labels, performing very well in the No labels. But the results in the Paragraph Suicide Attempt(SA) and the Paragraph Suicide Ideation(SI) class shown in figures 9 and 10 show a different story. While some categories performed extremely well as can be seen in figure 9 for the case of the Paragraph SA, where the Positive and Neutral labels performed very well, the Negative Unsure label performed poorly across all classification metrics used. Also, in the Paragraph SI prediction class, the model performed well in the Neutral sub label but poorly in identifying the Positive and Negative sub-labels. This brought the overall classification performance of these two models tremendously down. Table 1 shows the overall classification performance of the models across the three main classes.

Overall Performance Metrics of the various classes

	Classes	Precision	Recall	F-1
0	Paragraph Evidence(PEP)	0.87	0.89	0.88
1	Paragraph SA Prediction	0.62	0.64	0.63
2	Paragraph SI Prediction	0.61	0.69	0.64

Figure 14: Figure showing a table of the overall classification performance of the three class labels - Paragraph Evidence Prediction(PEP), Paragraph SA Prediction, Paragraph SI prediction from the ScAN and ScANNER research. The numbers data were gotten from Rawat et al. (2022)

One major reason for this bump in the research can be attributed to 'Hospital Stays' or what I might call 'relevance of the data during a time frame'. One must recall that this research is based on EHR taken from a hospital facility that had been annotated for further NLP processing. During the collation and subsequently annotation phase, the researchers failed to take into consideration that though a patient may have Suicidal Ideations for instance, form what point does it start and to what point has this suicidal ideations stopped? Also what is the duration of this particular suicidal ideation? How long does this patient have them and most importantly does the Health Practitioner delineate his/her notes between the different states of mind - having suicidal ideations and not having suicidal ideations. It seems that, irrelevant data may been accident and unconsciously been collated in the data collation phase and consequently annotated also. These flaw in the data engineering phase thereby affected the relevance and strength of the medRoBerta model in the two classes - Paragraph SA prediction and paragraph SI prediction as can be seen tin the overall performance metrics shown in Table 1.

The research sort to plug an essential hole when it comes to having data to perform NLP tasks with by creating their own publicly available corpus based on actual EHR. They also built a new model based on Google's Bert model called the medRoberta and in the Paragraph Evidence Prediction achieved a Precision Rate of 0.87, Recall of 0.89 and F-1 ratio of 0.88. Though more work would need to be done in the data engineering process for the ScAN algorithm, which the researchers clearly stated, ScAN and ScANER are useful algorithms that could help detect suicidal behaviour in Electronic Health Records(EHR).

2.3 LEARNING MODELS FOR SUICIDE PREDICTION FROM SOCIAL MEDIA POSTS by ning et al

Various researches have been taken in order to predict if an individual would commit suicide in 30 days and also in six months. This was the thematic underlings of the research by (Wang et al. 2021). In the research, they proposed various machine learning methods to be used to predict if an individual would commit suicide in either 30 days or in six months. Of the various machine learning methods used, their best method was the Deep Learning they created. Their data was gotten form the CLPsych 2021 shared task (Wang et al. 2021).

The goal of the research was to ;look at Social Media posts and from that data infer if the post had suicidal intonations or signs in them which would determine if the person would commit suicide in 30 days or in six months. They aimed to created a model with a forecasting ability to predict suicidal behaviour in 30 days to six months. This is commendable in the sense that, in this research they aimed to be able to find ahead of time if a Social Media post would lead to suicide in the future. This is diversion from current research in the area of using Natural Language Processing to predict if one would commit suicide as there the approach is usually using data from either confirmed suicide attempts or unconfirmed. But here, the data deals with a temporal dimension present in their techniques and also in the data used which is the CLPsych 2021.

CLPsych stands for Computational Linguistics and Clinical Psychology, an independent body that brings together researchers in Computational Linguistics and NLP, who are interested in creating research or have created research in these domains with a a focus on mental health and pyschology. Despite the relevance of the data, the classification metrics painted a different story and we can investigate why in the methodology.

For the word embeddings stage, two approaches were used:

1. Latent Features
2. Hand-Crafted Features

The latent features or feature extraction technique involved using the Doc2Vec approach. This feature extraction technique creates a numerical(or vectorized) representation of a group of words(or word depending on the parameter tuning) taken together as one unit. Since, computers cannot understand human-readable language, NLP practitioners turn each word to a single vector in space. And based on the distance function used(e.g euclidean distance metric), creates a word embedding of each word in the vector space.(may need to work on this part).

The second feature extraction used was the Hand-Crafted Features. Here the authors extracted features based on emotionsShao et al. (2019). Here, 12 emotion categories were generated which include pride, fear, sadness, anxiety, disgust, relief, shame, anger, interest, agreeableness and joy (Wang et al. 2021). This was quite inventive as indeed, to be able to detect emotions from text could be valuable feature in categorizing Social Media posts, especially in the context of the space - Social Media - whereby individuals are more prone to extreme expressions of various feelings to varying degrees. The authors went further to generate how intense each emotion was using a technique called the NRC Affect Intensity Lexicon (Mohammad 2017). An

in-depth look at the paper by Mohammad (2017) would be looked into later down this chapter. But in essence, what the NRC Affect Intensity Lexicon offers, is a way to attach a numerical number to four different emotional states - anger, fear, joy and sadness. The lower the number, the lower the intensity and the higher the number, the higher the intensity. This number ranges from 0 to 1. This was how the handcrafted features were extracted in the research and used as the second option for feature extraction in the paper.

Another handcrafted feature was Parts of Speech. This was done using Python's in-built Natural Language Processing Library called NLTK. The goal for using Parts of Speech as a feature was to be able to extract sentences with Personal Pronouns like I, me, mine, myself as there exists research that shows a positive and strong correlation between people that attempt suicide and their use of Personal Pronouns (Roubidoux 2012).

The last handcrafted feature extractor was based on a research paper titled 'Three-step theory(3ST): A New Theory of suicide rooted in the "Ideation-to-Action" Framework' (Klonsky & May 2015). In the paper, the authors presented a theory rooted in the ideation-to-action to be able to better understand suicidal patterns and more importantly to conceptualize the process flow from the though of committing suicide to actually committing suicide. As the former is at an early stage may not pose as much risk as the latter because in one case the individual can be saved and the suicide attempt prevented in the other the suicide has already happened(could be a success or failure). The Three Step Theory involved:

1. Suicidal Thoughts and Ideation arise from a combination of pain(psychological) and hopelessness (Klonsky & May 2015).

2. The degree to how connected the pain and hopelessness are (Klonsky & May 2015)

3. There is a progression from suicidal thoughts and ideation to suicidal actions/attempts and these are happen as a result of various acquired, dispositional and practical contributors to the ability to attempt suicide (Klonsky & May 2015)

The authors used the techniques and methodology present in the paper by Klonsky & May (2015) to build a dictionary of suicide related words and further embedding these words with the Word2Vec algorithm. Further processing was done to remove the stop-words(words that have no interpretative value to the meaning of a sentence eg 'the', 'a', 'an' that tend to clutter human language) and other special characters like hyphens, vernacular etc.

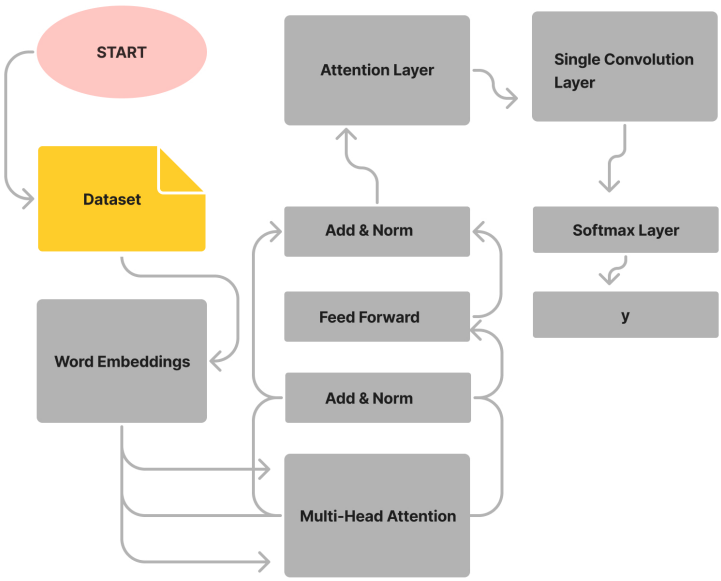

Figure 15: C-Attention Network Model by (Wang et al. 2021)

Figure 12 provides a visual summary for the C-Attention network algorithm used in the research. It begins with first calculating the word embeddings in the dataset, transforming each word to a vector, then processes it via a multi-head attention module that captures the relationships between the various vectors which is then followed by an attention layer followed then by a single convolution layer and finally the softmax function because this is a multi-class classification problem (Wang et al. 2021).

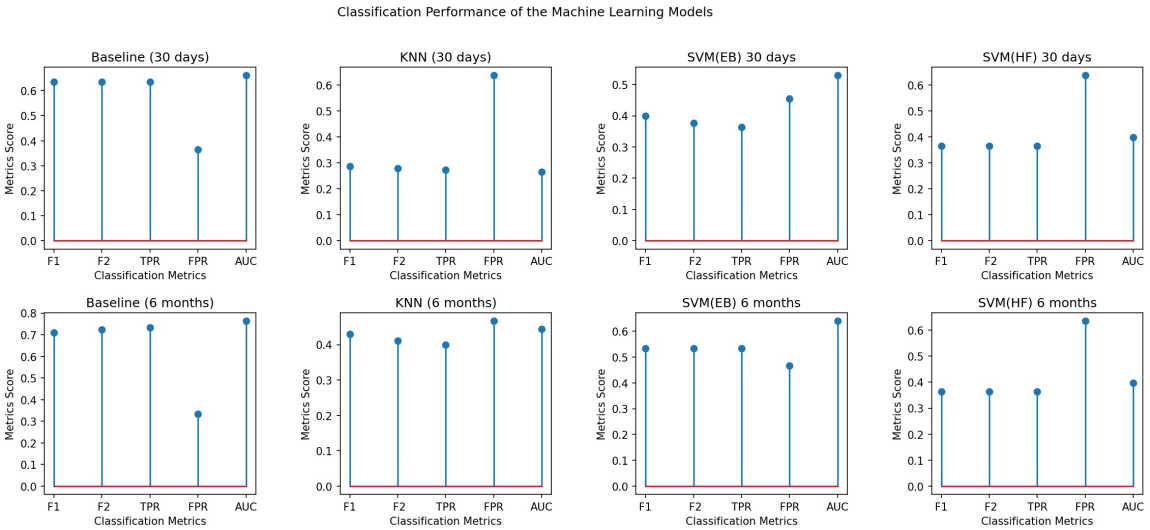

Figure 16: Classification Performance of ML Models 1. HF here stands for Hand Crafted Features

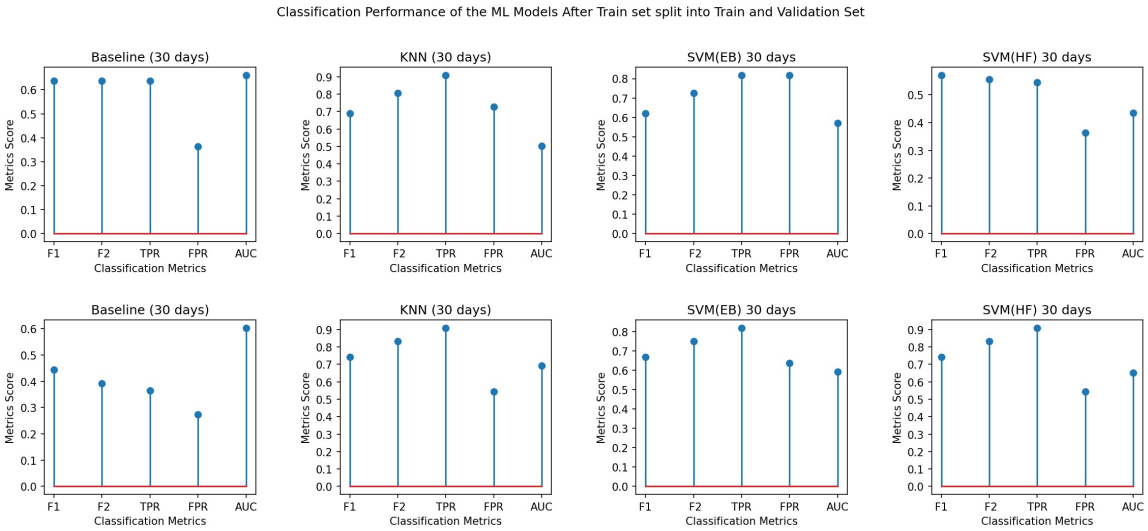

Figure 17: Classification Performance of ML Models after splitting the dataset into training set and validation set

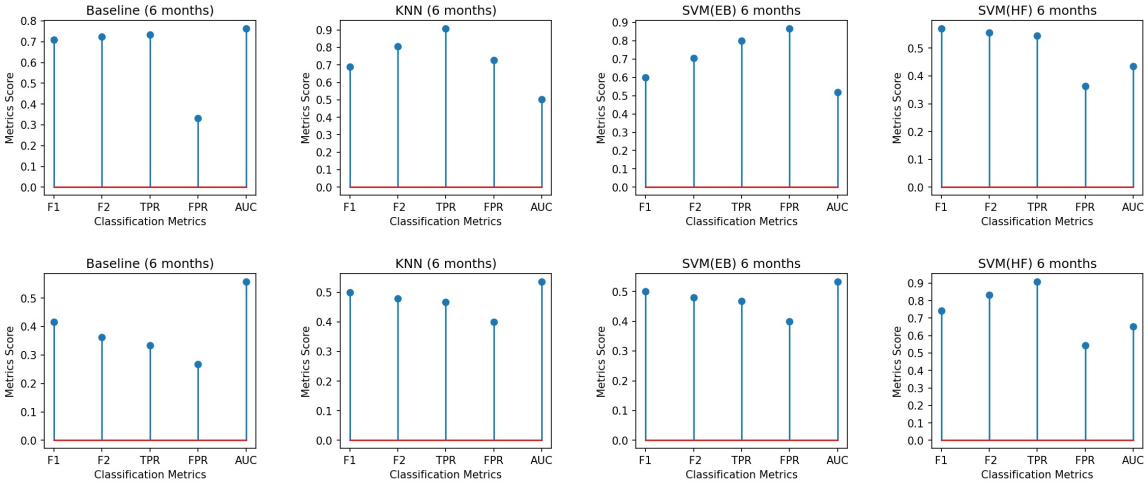

Figure 18: Classification Performance of ML Models after splitting the dataset into training set and validation set

It was explained in the research the reason for the poor classification performance of the various machine learning models as can be seen in figure 13; this happened as result of the fact that the models were over-trained on the training set. The researchers solved this problem by first dividing the training set into 80% train set and 20% validation set. Then the results improved as can be seen in figure 14 showing a few more ML models that were employed including - the C-Attention Deep Learning Model, Logistic Regression, Random Forest and Decision Trees. The best ML models according to performance is summarized in figure 15 below.

Best Performing ML Models

	ML Models	F1	F2	TPR	FPR	AUC
0	C-Att	0.690	0.806	0.909	0.727	0.504
1	KNN	0.741	0.833	0.909	0.545	0.694
2	SVM(EB)	0.741	0.833	0.909	0.545	0.653
3	RF	0.444	0.392	0.364	0.273	0.603

Figure 19: A table showing the ML Models that performed best in each of the Classification Metrics used in the research by Wang et al. (2021)

3 Research Methodology

3.1 Overview of Tools and Resources

Python was chosen as the Programming Language to build the various Document Similarity algorithms. Apart from a few of the native tools available in python, Python is also widely used in most Natural Language Processing tasks both for it's simplicity in syntax and it's strength in performing scripting tasks. Other tools and resources used in carrying out this research include:

1. NLTK Library: This is a Natural Language Processing Library built entirely in Python and used for performing Natural Language Processing tasks. A lot of the methods used in Tokenization and Word Embeddinggs as well as the various Preprocessing steps and tasks were done using various methods in this library.

2. Numpy: A Python Library package used for scientific computing and famous for its arrays - called Numpy Arrays. Numpy was chosen because of both it's convenient methods and also its speed. The ND-Array structure is known to be faster than in-built Python Lists. This is important especially in the deployment phase(the web application and the Twitter Bot) where speed is paramount.

3. Pandas: The Pandas library is used for analysing and processing data. It enables us to categorise and reshape data using the Pandas DataFrame class. Pandas is also known to work well with Matplotlib which is used for Data Visualization.

4. Matplotlib: This is a Python Data Visualization Library. It has a lot of Statistical plots, graphs and even animation. In this project, the library was used primarily to visualize the base data upon which rests the methodology for query search.

3.2 Twitter API Overview

The data used for building the Ensemble Model was gotten from the Twitter API Version 2 interface. The API is an advanced tool created for Developers to be used in carrying out research. There are four cadres to accessing data on the API:

1. Essential

2. Elevated

3. Elevated+

4. Academic Research

A link to the various levels of access for each cadre is available in the *work on this*. The Twitter API has what they call the Tweet Object. In the Tweet Object, there are various data available for mining and the one chosen for this project was the Search Data.

3.3 Twitter Search Tweets

The Search Data is further divided into two aspects:

1. Recent Search
2. Full-archive Search

The major differences between the two can be elicited from their names, the Recent Search endpoint allows one to access tweets over a definite time period of one week while the Full-archive Search has a time constraint dating back to the first tweet which was in March 2006.

The data for this project was fetched from the Full-archive Search with no time constraint. On the subject of constraint, it is important to briefly talk about the different parameters present in the Full-archive Search endpoint.

3.4 Twitter Search Tweet Object

The Full-archive Search endpoint has numerous parameters, there are briefly explained below:

1. query: This is the search query/term relevant to the project. In this research, a methodology was carried out and investigated in order to search for relevant query terms relating to suicide. These query terms were the result of data gotten from Electronic Health Records and are elaborated in the next section. The query parameter in the search endpoint is the only required parameter but automatically, the tweet object also returns a unique id. The query parameter has a maximum length of 1024 characters.
2. start_time: The start time from the tweets would be provided. It uses the UTC(Coordinated Universal Time) timestamp - YYYY-MM-DD H:mm:ss where:
    - YYYY - Year e.g 2010
    - MM - Month e.g July
    - DD - Day e.g 29
    - H - Hours
    - mm - minutes
    - ss - seconds
3. end_time: The latest and most recent date of which the tweets should be provided from, also using the UTC timestamp.
4. since_id: This parameter returns results with a tweet ID that is greater than the specified ID.
5. until_id: This does the opposite of the since_id parameter, it returns tweets with a tweet id that is less than the specified ID.
6. max_results: The maximum amount of results to be return by the tweet object from the search query. The maximum amount is 100 and the minimum is 10. By default the system returns the minimum amount.

7. next_token: This parameter is used to get the next page of results.

8. tweets.fields: a comma separated list of fields for the tweet object. The fields include id, text, attachments, entities, geo, lang, context_annotations etc. The default fields that return after a request to the endpoint are the id and text.

9. expansions: This is a comma-separated list of fields to expand.

10. media.fields: A comma separated list of fields for the media object. The default fields are the media_key and type.

11. place.fields: a comma separated list of fields for the place object. The default fields are id and full_name. Other fields include: country, contained_within, country_code, geo, name, place_type. This field in particular would be important in further research when building a model that is unique to a particular country. For this project however, a generalised model was attempted to be built that is not specific to any country.

12. poll.fields: comma separated listed of fields for the poll object. The pool object has to deal with the poll/voting functionality present in Twitter whereby users can create individual polls.

13. user.fields: comma separated list of fields for the user object. The default fields are: id, name, username.

For this project, the data used to build the model used only two parameters in the search query - the id, query and max_results. The purpose was to focus on the content of the text and for the model to be able to identify similarities between the search query and the Tweet Object in it's barest form. More on the advantages and disadvantages of this minimalist approach would be elaborated on in the section - Further Recommendations/Research in Chapter 5.


```
{
    "data": [
        {
            "id": "1555460252371484673",
            "text": "No worries one note lying gatekeeping Harm Reductionists. I got this whole thing with that empty nalaxone box done by myself.
                    You can all go fuck yourselves now. No seriously I don't want shit to do with y'all."
        }
    ],
    "meta": {
        "newest_id": "1555460252371484673",
        "oldest_id": "1555374662170906626",
        "result_count": 10,
        "next_token": "b26v89c19zqg8o3fpz5m80wk4cod41uri37ilgyb2cc59"
    }
}
```


Figure 20: A returned request sample of a Search Query in the Twitter API V2 using Postman. The remaining 9 tweets have been cut off so the image could fit the page.

The figure above shows a sample of the returned request from the Twitter API using Postman. As can be seen in the image, it returns a Javascript Object Notation(JSON). In Postman, one can transform this JSON to XML, HTML, or even just Text. The JSON format was used for the data. The figure shows the two default parameters - 'ID' and 'text'. More information is contained in the meta key, it shows various information about the get request including the 'result_count' - from the max_count field and other information like the 'next_token' etc.

3.5 Background for Querying Data

Querying the data for related terms like 'suicide' or 'death' cannot be an effective methodology in achieving our aims and objectives for this research. That methodology would create a model which would be good at identifying the keyword 'suicide'. But the purpose of this research is not to identify texts with the keyword of 'suicide' in them but rather to be able to detect in text, what is known as Affect in Natural Language Processing.

Affect can be loosely defined as words that contain different feelings, emotions, and attitudes (Mohammad 2017). This is especially important as this research seeks to build a model that would be able to recognise particular affects present in tweets from users. There exists various Sentiment Lexicons that contains a dictionary of various word affects ranging from fear to anger to joy to sadness to guilt to different emotions. Each Sentiment Lexicon have different in-built affect.

3.6 Sentiment Lexicons

While crafting the methodology for querying the data from Twitter API endpoint, the Affect Intensity Lexicon created by (Mohammad 2017) was used. This Affect Intensity Lexicon was a combination of various Sentiment Lexicons. They include:

1. AFN: The AFINN Sentiment Lexicon contains a list of English words and terms that have been manually rated for valence with an integer between -5(negative) and +5(positive).This lexicon was initially built with obscene language and words but was subsequently expanded to include terms sourced from Twitter, internet slang dictionaries and affected word lists (Mohammad 2017).

2. EMO: The NRC Emotion Lexicon also known as Emolex, is a list of English language words and their relationships grouped into eight different affect categories - anger, fear, anticipation, trust, surprise, sadness, joy and disgust and two sentiments(negative and positive) for each of the affect categories.

3. LWC: The Linguistic Inquiry and Word Count(LWC) is a tool used in analyzing word use. It can be used to study single individuals, group of individuals or even larger sections of the society. In the research of (Mohammad 2017), it was used in creating meaningful affect and psychological categories. It is important to note that trhe LWC tool was originally created to analyse an individual's narratives so as to extract valuable insights from the writer's pyschology and further understand the correlation between that individual's word use and health outcomes.

4. OPN: The Opinion Lexicon is a publicly available, free to use lexicon made up of 10 classes in general - 5 positive and 5 negative. The positive classes include:

   (a) Positive Noun e.g "attraction", "astonishment", "awe"

   (b) Positive Adjective e.g "jolly", "irreplaceable", "jaw-dropping"

   (c) Positive Adverb e.g "gladly", "fairly", "precious"

(d) Positive Verb e.g "cherish", "empathize", "heal"

(e) Positive Phrase e.g "I love flowers."

While the negative classes include:

(a) Negative Noun e.g "harm", "ache", "hurt"

(b) Negative Adjective e.g "abused", "anxious", "depressed"

(c) Negative Adverb e.g "miserably", "worse", "angrily"

(d) Negative Verb e.g "assault", "choke", "cry"

(e) Negative Phrase e.g "I cried till my eyes were swollen."

It uses a polarity score whereby a word would have a desirable state e.g "great", "good" - have a positive polarity while words with an undesirable state have negative polarity e.g "bad", "awful".

5. PAT: is an NLP package that has a component that can be used for sentiment analysis. It uses a lexicon, which is a sub-component of the large database of English , containing Nouns, verbs, adjectives and adverbs grouped together into lists of various cognitive similarities with each each expressing a unique concept called Wordnet. Each word has a sentiment score between -1(negative) to 1(positive). PAT processes the context of each word from its negation(e.g not worried) and its adverbial modification (e.g very worried) and provides a single score for it's polarity. The Pattern Lexicon was applied in the research by (Mohammad 2017), removing all the words with a neutral polarity score(0).

6. SWN: The SentiWordNet is a lexical resource for Sentiment Classification. It assigns to each synset of WordNet three sentiment scores: positivity, negativity and objectivity. For each word in the lexicon, the higher of the two scores was retained as the word's sentiment and in the situation whereby the two scores were the same, a random sentiment was chosen. For simplicity sake, only single-word entries were chosen from this lexicon.

3.7 Methodology for Querying Data

The Affect Intensity Lexicon (Mohammad 2017) was used in the methodology for deriving the keywords used to mine data from the Twitter API Version 2 Full-Archive Search endpoint. Figure() below gives n overview of the data structure and information about the Affect Intensity Lexicon.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Unnamed: 0    195 non-null    int64
 1   word          195 non-null    object
 2   case_freq     195 non-null    int64
 3   control_freq  195 non-null    float64
 4   mwu_u         195 non-null    int64
 5   mwu_p         195 non-null    float64
 6   freq_ratio    195 non-null    float64
 7   freq_diff     195 non-null    float64
 8   afinn         32 non-null     object
 9   emolex        41 non-null     object
 10  liwc2015      22 non-null     object
 11  opinion       25 non-null     object
 12  pattern       9 non-null      object
 13  swn           86 non-null     object
dtypes: float64(4), int64(3), object(7)
memory usage: 21.5+ KB
```

Figure 21: An image showing the various information about the Affect Intensity Lexicon

The figure above provides a summary of essential information about the Affect Intensity Lexicon Dataset like the amount of entries, number of fields, data types of the different fields etc. This image gives us a picture of how the data is structured in the Affect Intensity Lexicon(AIL) dataset.

Now that we are aware of the basic structure of the fields, rows and data types present in the AIL dataset, it is also pertinent to have a closer look at the data before doing any statistical or computing analysis with the data. This would provide us with much better insight into the type of data allowed in the columns for instance, the range of values and most importantly the fields that would be relevant for this research. Figure() shows the head of the data(i.e. the first five rows) and figure() shows the tail of the data(the last five rows).

	Unnamed: 0	word	case_freq	control_freq	mwu_u	mwu_p	freq_ratio	freq_diff	afinn	emolex	liwc2015	opinion	pattern	swn
0	1	self	4279	2217.642490	4038260844	0.0	1.929302	2060.860661	NaN	NaN	NaN	NaN	NaN	NaN
1	2	harm	2916	1242.740515	4058706079	0.0	2.346568	1673.434190	negative	negative	negative	negative	NaN	negative
2	3	ward	5555	3957.645900	4095271325	0.0	1.403544	1597.086158	NaN	NaN	NaN	NaN	NaN	negative
3	4	overdose	1717	324.209649	4191119922	0.0	5.295901	1392.772579	NaN	negative	NaN	NaN	NaN	NaN
4	5	staff	5670	4280.643267	4182314937	0.0	1.324579	1389.404933	NaN	NaN	NaN	NaN	NaN	NaN

Figure 22: An image showing the first 5 elements(column and row-wise) of the AIL data.

	Unnamed: 0	word	case_freq	control_freq	mwu_u	mwu_p	freq_ratio	freq_diff	afinn	emolex	liwc2015	opinion	pattern	swn
190	3376	health	2282	3053.283579	4449762431	1.990000e-30	0.747469	-771.049528	NaN	NaN	NaN	NaN	NaN	positive
191	3377	medication	3757	4773.890478	4388371377	5.320000e-52	0.786893	-1017.350878	NaN	NaN	NaN	NaN	NaN	positive
192	3378	appointment	1584	2708.010532	4335412955	2.530000e-162	0.584761	-1124.470665	NaN	NaN	NaN	NaN	NaN	NaN
193	3379	mr	1198	2335.991514	4483837710	1.820000e-28	0.512793	-1138.110803	NaN	NaN	NaN	NaN	NaN	NaN
194	3380	mental	3093	4335.044419	4277865863	1.770000e-182	0.713383	-1242.496835	NaN	NaN	NaN	NaN	negative	positive

Figure 23: An image showing the last 5 elements(column and row-wise) of the AIL data.

From a look at the images above, we are shown a better literal perspective of the data contained in the AIL dataset. Since the purpose for using this dataset is to mine the relevant keywords from the Electronic Health Records and other relevant Sentiment Lexicons that were used to build the AIL dataset, the columns that would interest this research the most would be both the word and case_freq column.

The choices for the word column is that it provides us with the necessary information for the relevant words to be used in querying the data and for that of case_freq is know how relevant each word is according to their number of occurrences. The more times a word appears in various EHR's the more relevant that word is and vice-versa.

Hence, we would need a methodology for engineering the data to be able to provide us with this insight. Before providing the methodology for this, let us isolate the columns to be able to have a clearer perspective. The figure below provides this perspective:

```
            word  case_frequency
0           self            4279
1           harm            2916
2           ward            5555
3       overdose            1717
4          staff            5670
..           ...             ...
190       health            2282
191   medication            3757
192  appointment            1584
193           mr            1198
194       mental            3093

[195 rows x 2 columns]
```

Figure 24: An image isolating the relevant columns/fields for the Query Methodology - word and case_freq.

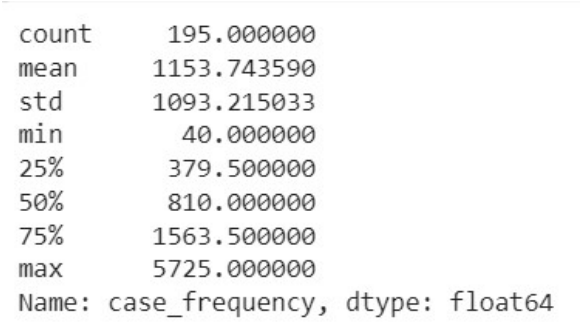

Figure 25: An image showing the Descriptive Statistics of the case_freq column.

The figure above shows a summary of the Descriptive Statistics of the case_freq column. From this data, we can choose an appropriate threshold to find the most occurring words in the AIL data. For this research words with a case frequency in the 75th percentile and above were used.

After isolating words in the 75th percentile and above, next was to select those words with the corresponding 75th percentile case frequencies. There were a total of 48 data entries that fell into this percentile. The algorithm for this is shown below as well as the table of all the top 48 words.

Algorithm 1 Algorithm for Engineering the ALI word and case_freq data column

Require: $words = []$
  for $index, value$ in $column_name$ **do**
    for $index_2, value_2$ to $column_name_2$ **do**
      if $value > percentile$ **then**
        if $index = index_2$ **then**
          $words.append(value)$

	top words	top frequencies
0	self	4279
1	harm	2916
2	ward	5555
3	overdose	1717
4	staff	5670
5	suicidal	2072
6	said	5725
7	alcohol	2276
8	risk	3081
9	admission	2333
10	thoughts	2313
11	go	2356
12	discharge	1813
13	home	3316
14	plan	4347
15	hospital	3005
16	low	2189
17	call	1915
18	stated	1945
19	asked	2341
20	informed	1821
21	feeling	1844
22	history	2251
23	would	5725
24	felt	1744

Figure 26

25	back	2346
26	agreed	2282
27	feels	1926
28	patient	2363
29	leave	2827
30	mood	3421
31	told	1631
32	could	2448
33	going	1636
34	today	2357
35	morning	1577
36	currently	1659
37	reported	3071
38	night	2425
39	treatment	1593
40	assessment	2643
41	care	2863
42	attended	1621
43	state	1640
44	health	2282
45	medication	3757
46	appointment	1584
47	mental	3093

Figure 27: A table showing the 75th Percentile of the words according to their frequency in the AIL Dataset.

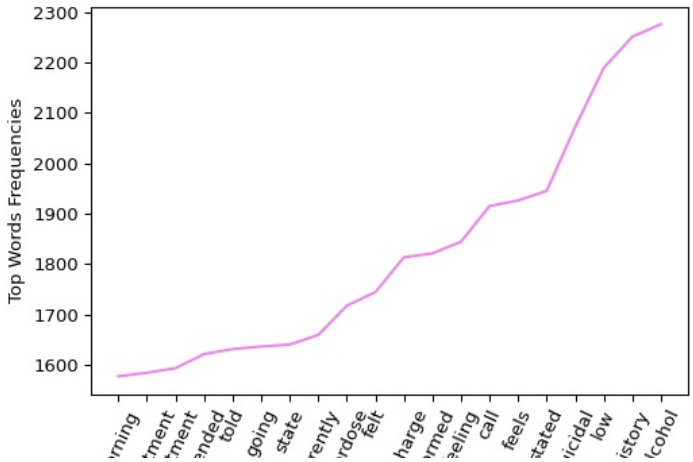

Figure 28: This line plot helps us to understand how the data values fluctuate.

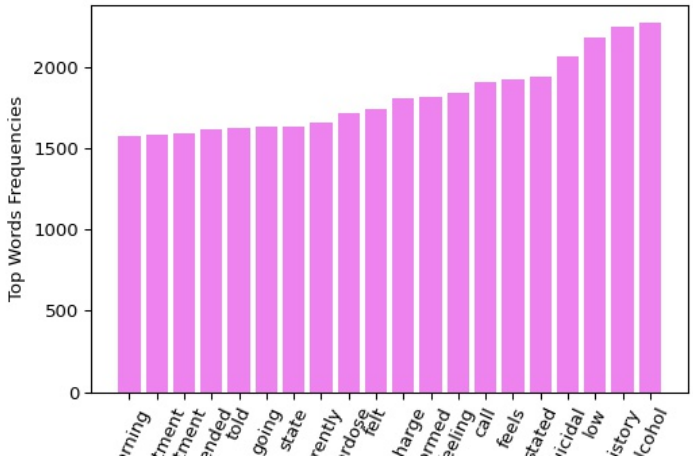

Figure 29: This vertical bar chart helps us understand the spread of the words in the 75th percentile.

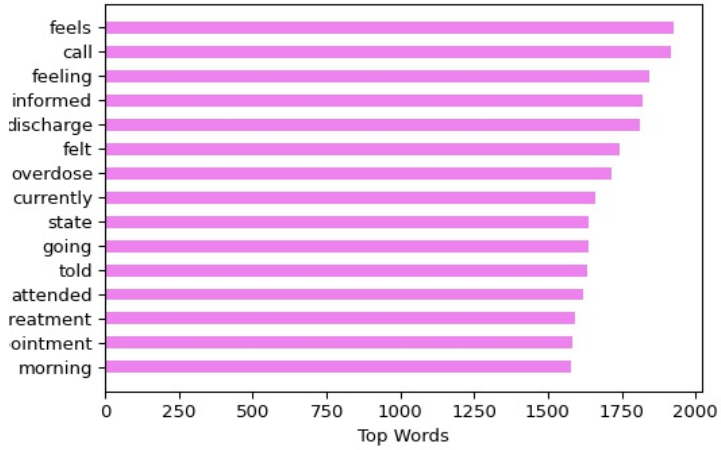

Figure 30: This horizontal bar chart helps us understand the value range(using case frequencies) of each word.

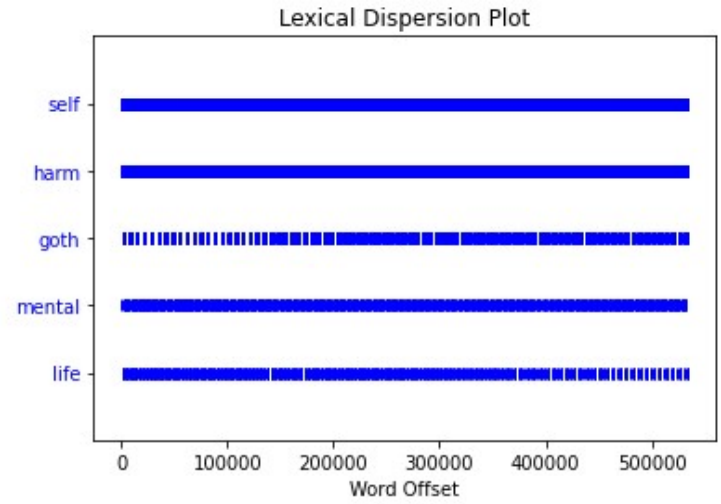

Figure 31: A Lexical Dispersion Plot showing the top words in the engineered data and their spread in the corpus.

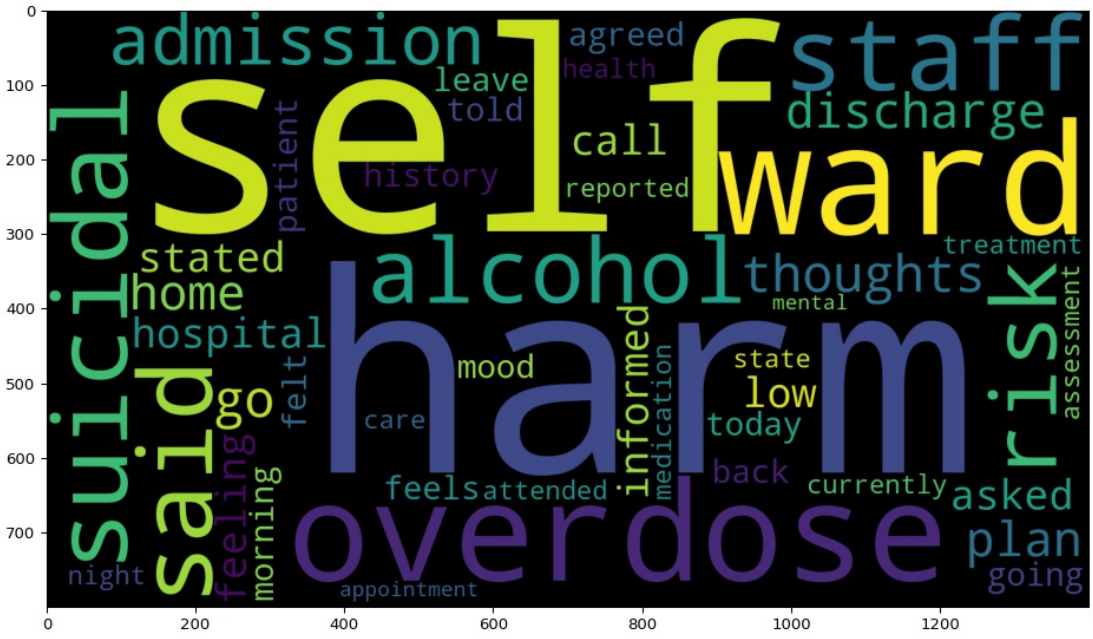

Figure 32: This Word Cloud helps us to better visualize the weight and frequency of each word from the engineered AIL dataset.

The flow chart provides a Visual Summary of the methodology used for selecting keywords that were further used for querying the Twitter Developer API Full-Archive Search endpoint:

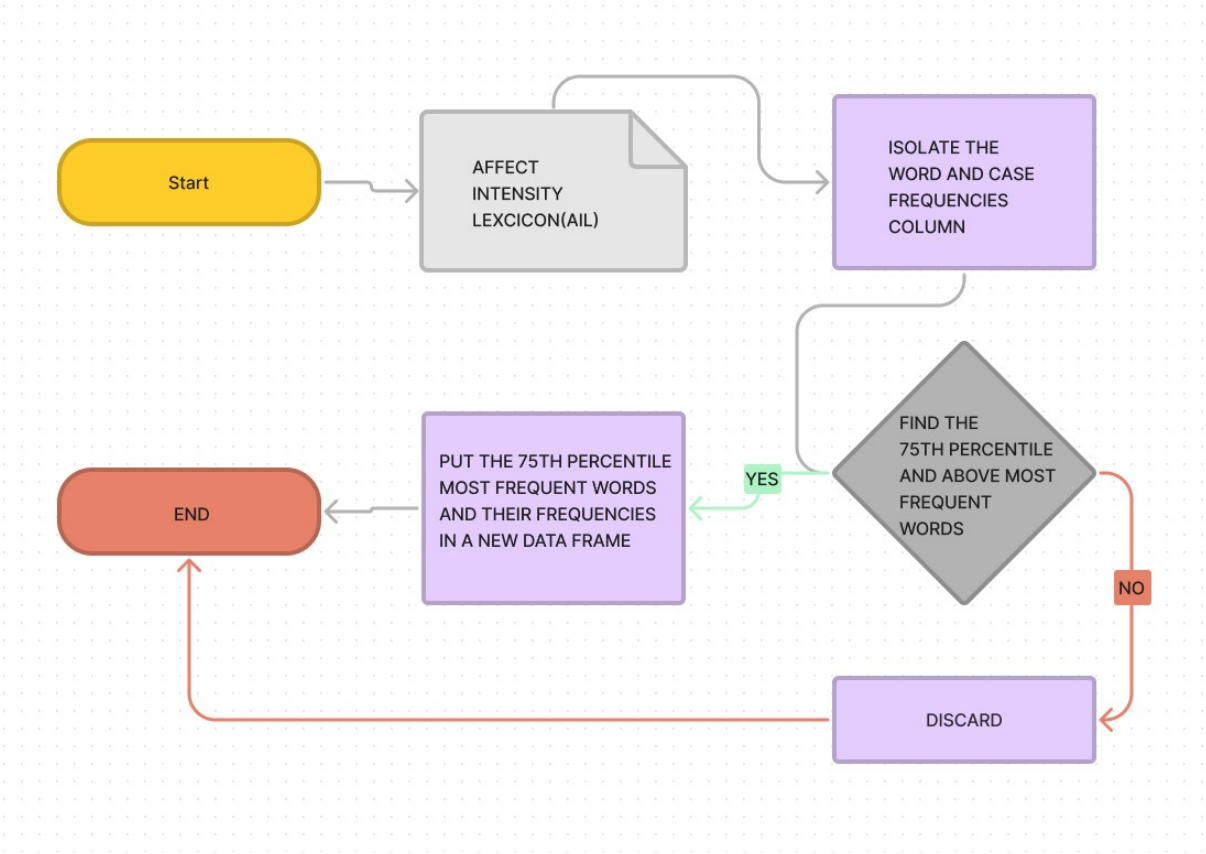

Figure 33: A flow chart showing the algorithm/methodology for selecting the keywords that would be used to query the Twitter API Full-Archive Search endpoint.

3.8 Algorithm to Clean Twitter Data - Data Preprocessing I

A look at figure 20 in the Twitter Search Tweet Object sub-section, we can see that the Twitter API returns a query filled with extra characters and text that are not important to the research. And in some cases, would tilt our data towards a particular spectrum for example the heavy presence of the escape character - ”. This means that it is paramount to first clean the data before building the model.

An algorithm was then created, specifically for the task of cleaning the data request sent from the Twitter API that would be used to clean the data for the models and also when in live production, used to clean text. The algorithm is shown below:

Algorithm 2 Algorithm for cleaning the Twitter API Full Archive Search Query Data

Require: $data = ''$

$open\ (text_file, 'read')\ as\ object$
$datum = object.read()$
for $value$ in $datum$ **do**
if $\#\ in\ datum$ **then**
$value =$ ""
end if
if $\backslash or\ \backslash\backslash or\ \backslash\backslash n\ or\ \backslash\backslash n\backslash\backslash n\ in\ datum$ **then**
$value =$ ""
end if
if $\backslash\backslash n\backslash\backslash n\ in\ datum$ **then**
$value =$ ""
end if
if $\backslash n\backslash n\ in\ datum$ **then**
$value =$ ""
end if
if $\{\ or\ \}\ or : or\ \_\ or\ @\ or\ [\ or\ \backslash or\ ]\ or\ 0-9\ or\ ,\ or\ \%\ or\ \&\ or\ *\ or\ ''\ or\ ''\ or\ ?\ or\ !\ or\ -$ $in\ datum$ **then**
$value =$ ""
end if
if "id" $in\ datum$ **then**
$value =''$
end if
if "$text$" $in\ datum$ **then**
$value =''$
end if
if "RT" $in\ datum$ **then**
$value =''$
end if
if "␣␣" $in\ datum$ **then**
$value =''$
end if
$data = datum$

What the Cleaning Twitter Data Algorithm above those is that it removes every extra characters, unnecessary white space and text/words from the return query result, thereby making the data cleaner, and more importantly getting to the bone of the real intent behind the tweet. This algorithm is only for the default parameters - id and text, other parameters like media, polls etc would need a different cleaning algorithm to handle those case scenarios.

3.9 SWEETS CORPORA

The Corpora has a total of 1,073,595 words. The Word Cloud below provides a Visual summary of the most frequent words.

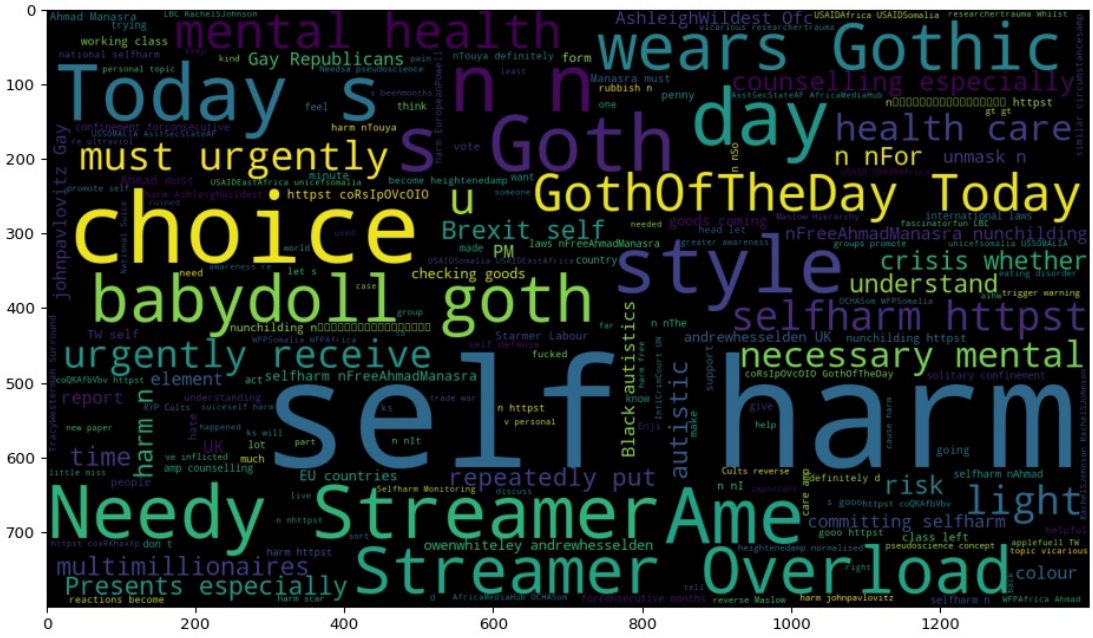

Figure 34: Corpora Word Cloud

The figure below shows the various words on the horizontal axis(x) and their selected word counts on the vertical axis(y).

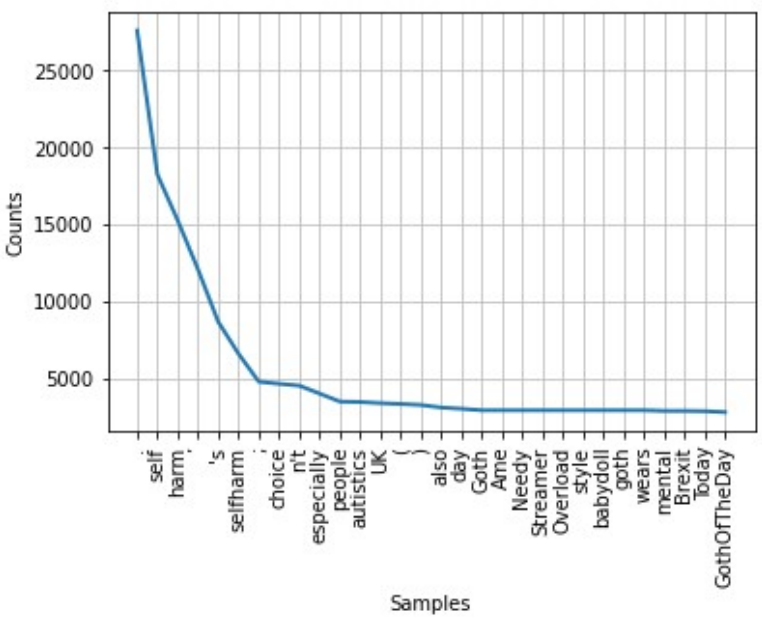

Figure 35: Corpora Word Tokens Frequency of 30 words

The process for creating the Corpora was by using the methodology involved in mining the most frequent words/tokens in the AIL Lexicon. A process of using the top word combinations like "self-harm" etc was used to extract data from the Twitter API Full-Archive Search endpoint. At this point, Stemming was already applied to the tokens to remove different lexical variations of words to avoid unnecessary repetition in the Corpora. This is elaborated more later in this chapter.

Further processing of the Corpora, extracting Collocations, Bigrams, Trigrams, and even further cleaning would be elaborated in the Analysis section - Chapter 4.

3.10 SWEETS CORPUS

The Corpus, a smaller version of the Corpora, has a total of 402,777 words. The figure below is a Word Cloud showing the Visual summary of the frequency of each word and figure() shows the frequency on the y-axis and each word on the x-axis for further examination and understanding.

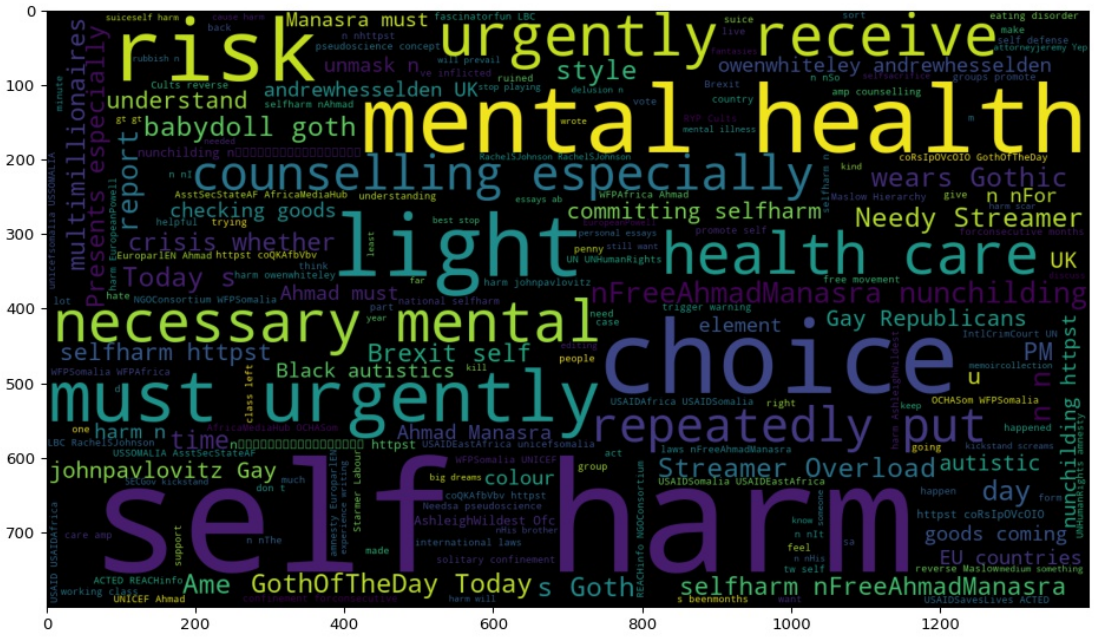

Figure 36: Corpus Word Cloud

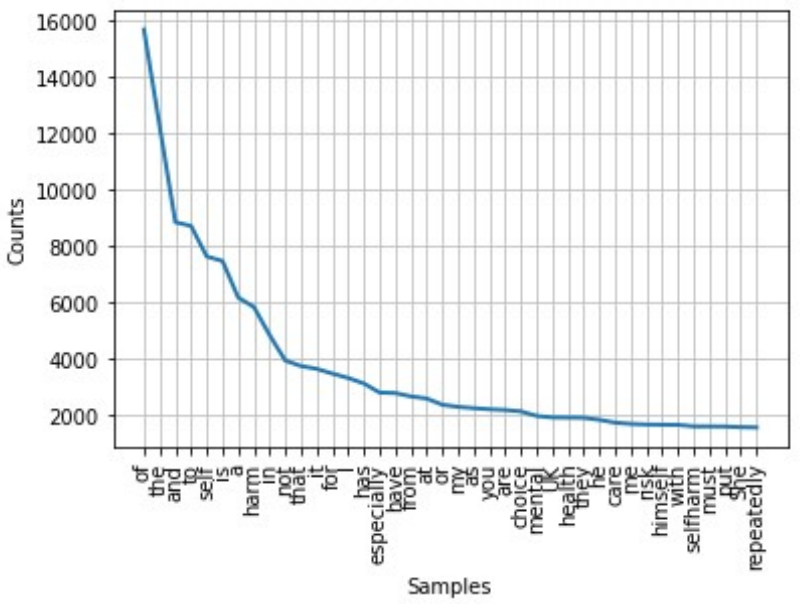

Figure 37: Corpus Word Tokens Frequency of 30 words

A Corpus was created so has to enable both I and other researchers to have access to a smaller version of Twitter Word tokens relevant to research on Suicide. A good use-case would be elaborated in Chapter 4 when it came to training the Corpus using the Bert Model, because

of the size of the Corpora and even the Corpus, a further smaller version of the Corpora had to be created in order to train the model on a CPU. On a GPU this would not be a debate but computational costs and complexity begin to incur debt in the deployment phase.

The limitation mentioned above would be further examined in both Chapters 4 and 5 of this research work.

3.11 SWEETS MINI

A mini version of the Corpora was created for purposes of Machine Learning Operations and Deployment. This version of the Corpora contains 235,392 words. The figures below shows the Word Cloud and Frequency Distribution of the different words in the Corpora Mini.

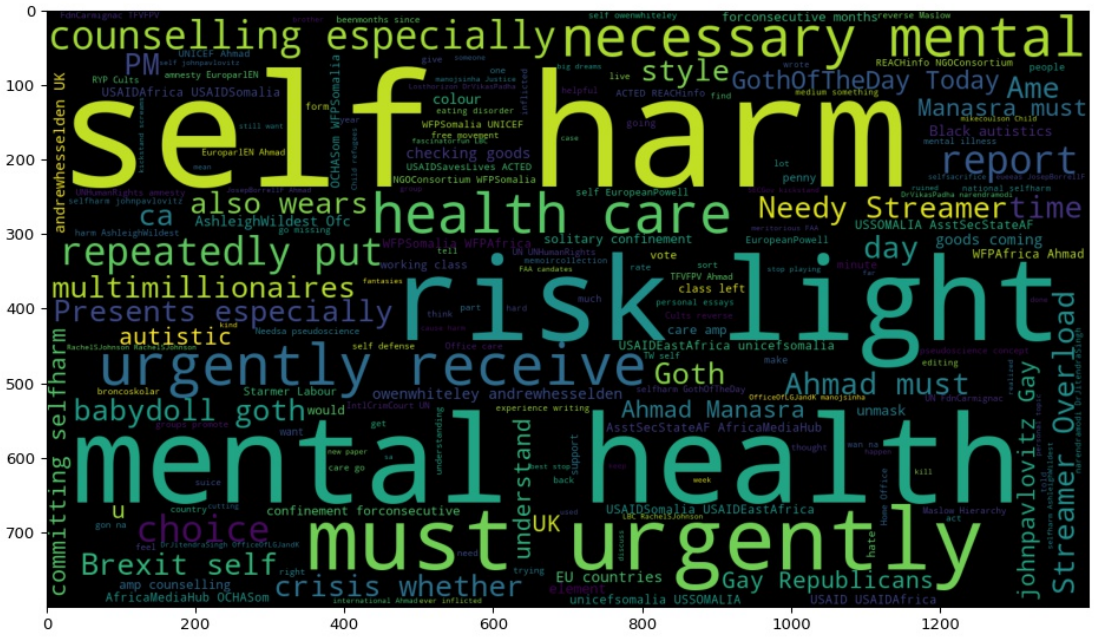

Figure 38: MINI Word Cloud

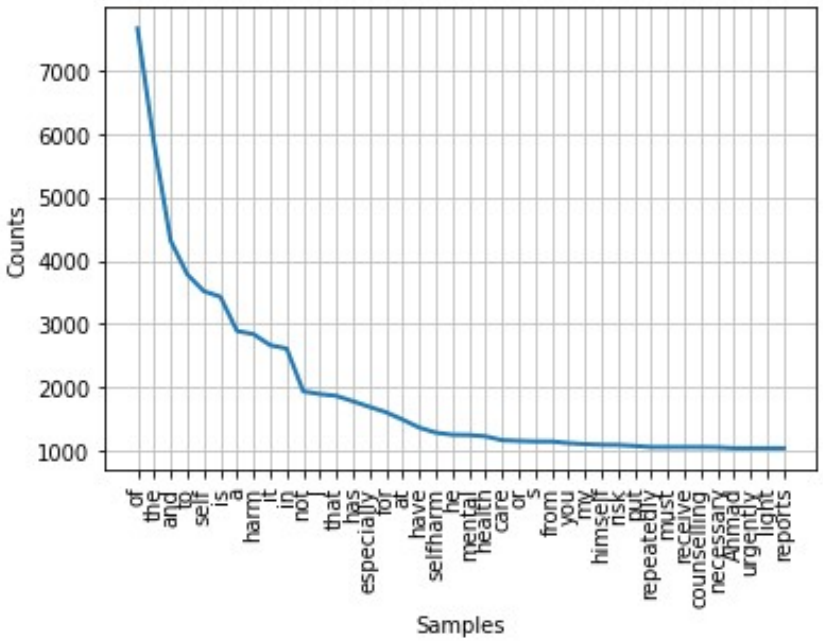

Figure 39: MINI Word Tokens Frequency of 40 words

3.12 Tokenization and Word Embeddings

This can be defined as the segmentation of text into small, linguistic units such as words, punctuation, numbers, alpha-numerics etc (Mikheev 2003). It can also be defined as the process of dividing words into separate units, called tokens and in the process discarding certain characters e.g punctuation. The usefulness of tokenization of text is that it helps when one wants to create a custom corpus and it also assists in inferring relationships between certain words or group of words in a text.

$$tokens = nltk.word\_tokenize((corpus), width = (integer), compact = (Boolean))$$

The above Tokenization equation is for creating single word tokens. While creating single word tokens is useful and practical, in this research, I have decided to create two more other tokenization techniques and Word Embeddings. There are:

- Sentence Tokenization
- Tweet Tokenization

Sentence Tokenization has to deal with creating tokens or turning sentences into tokens. While Word Tokenization turns each word tro an integer in the vector space, Sentence Tokenization turns each sentence to an integer in the vector space. The nltk library has a method called sent_tokenize in the tokenize class that can handle Sentence Tokenization.

Tweet Tokenization: while carrying out research on the Word Tokenization class in the nltk library, I came across the TweetTokenizer class which accepts as an input twitter style data and tokenizes each word. This class was primarily built to turn tweets into tokens and would be a good feature for the Feature Engineering process.

This tokenizer class is a tweet-aware tokenizer that has been designed and created to be flexible and easy to adapt to new domains e.g Twitter and other Social Media. It has a few parameters, I think it is important to briefly elaborate on them below:

1. preserve_case: the default for this parameter is True. If it is set to False, it would change all the letters in the string argument to lower case.

2. reduce_len: the default of this parameter is False. It has to deal with replacing the repeated character sequences of length 3 or greater with sequences of length 3.

3. strip_handles: the default is set to False. If True, it would remove the user handles. This would have been done in the Twitter Clean Algorithm set earlier.

4. match_phone_numbers: the default of this parameter is set to True. If False, it would not look for phone numbers in the string argument.

3.13 Tokens Feature Engineering

The Word, Sentence and Tweet Tokens were all used as features for the Document Similarity Algorithms. The reason for this is to try to see if there are difference in similarity when the words are tokenized by each word or sentence or by nltk's purpose-built Tweet Tokenizer. Also, for one of the Document Similarity Algorithms, BERT, it requires the words to be sentence tokenized as opposed to word tokenized.

Using the MINI data, lets have some Visual and Statistical samples of these different features so we can see their differences and similarities(if any).

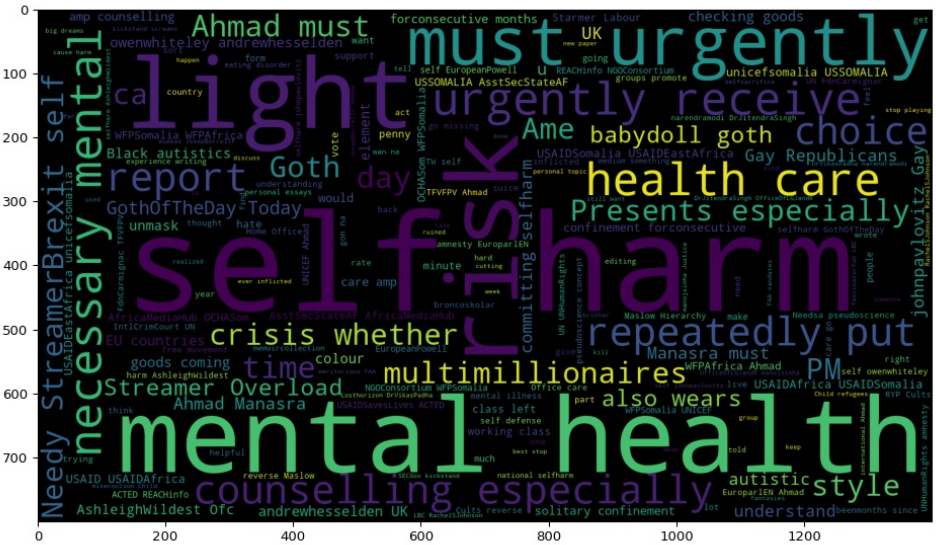

Figure 40: MINI Word Cloud using the Word Tokens

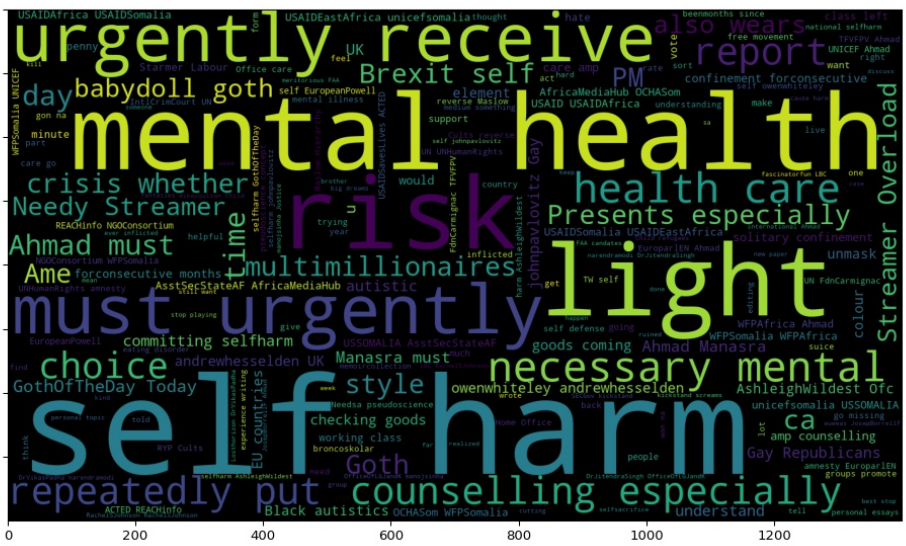

Figure 41: MINI Word Cloud using Sentence Tokens

Figure 42: MINI Word Cloud using Tweet Tokens

We can see in the Word Clouds above that both the Word and Tweet Tokens arrive at similar results. This could be attributed to the cleaning algorithm developed earlier, whereby each data

mined from the Twitter API full-archive search endpoint has already been cleaned appropriately.

The Research Methodology and Analysis for this project are somewhat intertwined. Because this data is domain specific and even context-specific, some of the results were shown in the Methodology chapter like the descriptive statistics of the base data and that of the corpora, corpus and mini as well as the various Word Clouds shown in the chapter.

In this chapter, some of this information would be referred to but not repeated. The focus of this chapter would be about analysing the corpora, corpus and mini as well as the individual algorithms(Jaccard, TF-IDF, ENSEMBLE, and Bert) as well as the Ensemble Techniques used in building the model from the aspect of the model's efficiency and effectiveness both with the context data - User Tweets and even non-context data - Electronic Health Records.

Two main techniques for measuring a model's performance would be used:

1. True Positive Rate(TPR)
2. True Negative Rate(TNR)
3. Accuracy Score
4. Recall

3.14 Common Contexts

While the similar method in nltk helps us understand words that appear in a similar range of contexts, one needs to be able to know the context between two words. This is especially important as if you recall, the base document - MINI - has been transformed into tokens. Which means that each word has been separated and are now individuals whereas some words like 'self-harm', 'mental health' offer more context in pairs.

3.15 Stop Words

Stop Words are those words in the English Language that offer little to no lexical and in some cases contextual meaning to a sentence. e.g most prepositions and definite and indefinite articles like 'the', 'a', 'an'. Conjunctions like 'and', 'but' etc These words are often ubiquitous in sentences but when it comes to a lot of Natural Language Processing tasks, there offer little to no meaning in analysis. According to (Gerlach et al. 2019) stopwords can be loosely defined as the removal of uninformative words. Hence, it is a good idea to isolate and remove them in order to make our models lighter in the amount of relevant word embeddings/tokens.

In the nltk library, there is a class called Corpus which has a method called 'stopwords'. This method was used to remove the stopwords in the Corpora, Corpus and MINI. It would also be used in the deployment phase to remove live samples of User Tweets when doing the comparison between the user data and the ensemble or one of the Document Similarity Algorithm models.

3.16 Document Similarity

Finding the similarity between two or more documents is an important step and utility in a lot of Natural Language Processing tasks. Document Similarity is used in various domains and its

purposes range from spotting plagiarism between two or more documents, finding the similarities between documents, recommending similar books and articles etc.

In this research, the aim of using Document Similarity is mainly to identify the level of similarity between a user's tweets and the model built from a series of user's tweets that were matched or mined from keywords fetched from the Affect Intensity Lexicon(AIL). The methodology for using Document Similarity in this research as explained in the previous sentence is shown in the flow chart below:

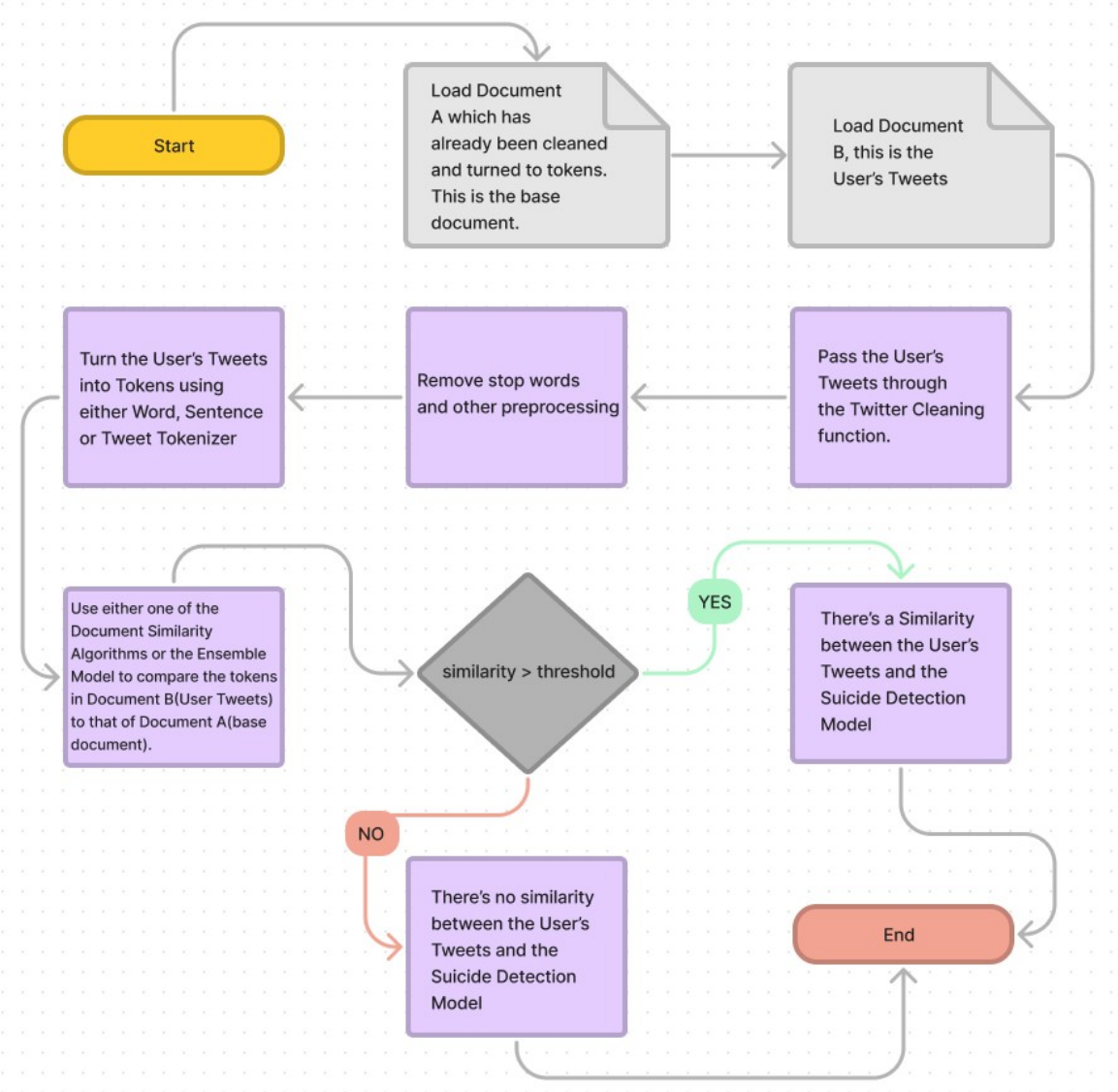

Figure 43: Suicide Detection Algorithm Flow Chart

In order to make machines understand the similarities between different texts, each of the texts must first be converted to a language the machines would understand. This is where the Tokenization step comes in, whereby each word, depending on the tokenization method used e.g it could be sentence tokenization as elaborated in a previous subsection, is turned into a token or vector. In this form, the machines can understand the words, after each document is turned into a series of tokens, the machine then compares the two different tokens from the two different documents and measures their similarity or differences. In this research, a certain threshold

would have to be chosen, when say, if Document A and Document B have 42% similarity score, then there are either similar or less similar. The lower the percentage the more similar the documents are and vice-versa.

3.17 Document Similarity Algorithms

Three Document Similarities were implemented in this project. Each of them increasing in their complexity and awareness of context in each documents. Then at the end, all three of them were combined into an Ensemble Model. The three Algorithms used in this research were:

1. Jaccard
2. TF-IDF
3. ENSEMBLE
4. BERT

3.18 Jaccard

If one were two take the total amount of words in two documents, then also take the amount of words that over-lap in both documents, then find the ratio of words that overlap : total amount of words in any documents, we would get the Jaccard Index. This simple yet intuitive algorithm was developed by Swiss Botanist, Paul Jaccard.

In this research, the Jaccard Index is used to measure the similarity between two documents. In order to compute the Jaccard Index, the stop words have to be removed first, hence we would be left with only relevant, contextual words in any document. Let's have an example and compute the Jaccard Index then go on to represent this algorithm visually using a Venn Diagram.

Suppose we have Document A: "I really wanna do self harm but why am i so scared"

Then Document B: " All of this except I don't have Netflix. Watching this is self-harm for those of us too squeamish to watch Casualty."

Removing the Stop Words from Document A we have: "really wanna self harm scared"

And doing likewise for Document B: "except Netflix Watching self-harm us squeamish watch Casualty"

Words that over-lap are "self" and "harm" hence we have:

$$JaccardIndex = WordsthatOverlap/TotalNumberofWordsinAnyDocument$$

The Jaccard Index for the above example is calculated as:

$$JaccardIndex = 2/12 = 0.1667$$

The Venn Diagram below shows the relationship between the words in Document A and Document B, in the process, finding the similarity between the two documents, using the Jaccard Index would be 0.1667% as calculated previously.

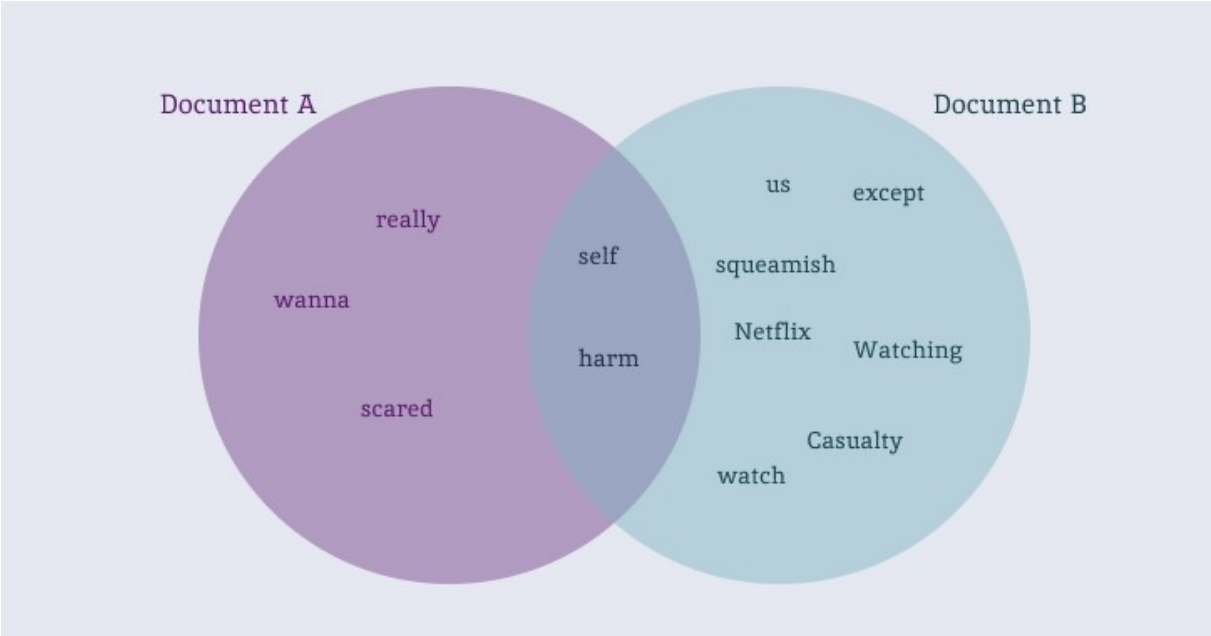

Figure 44: A Venn Diagram providing further Visual Illustration of how the Jaccard Index is computed.

3.19 TF-IDF

TF-IDF is a popular algorithm used in extracting features from a corpus. It is a technique used to quantify words in a text document. It differs from the also popular Bag of Words(BOW) model in that while the BOW model measures the frequency of a word no matter the amount of times, which could lead to redundancy especially when it comes to nouns e.g a Guardian article about Amy Winehouse would definitely mention Amy Winehouse n-number of times; TF-IDF on the other hand measures the relevancy of words in the document.

3.20 Term Frequency

The acronym TF-IDF stands for Term Frequency – Inverse Document Frequency. Term Frequency(TF) has to deal with measuring how frequent a particular word occurs in a given corpus. This would inevitably depend on the length of the document. For example, a conjunction like 'and' would appear more in a document with 10,000words than in a document with 1,000words; what this means is that it is a good idea to normalise this frequency by dividing the TF by the total number of words. Hence we have:

$$TF(t,d) = countoftind/numberofwordsind$$

where:
t = term(word)

d = document(set of words)

The essence of Feature Extraction is due to the fact that computers cannot process human language like humans do. In order for computers to be able to process language, they must be converted to matrices or vectors. This is one of the use-cases for Feature Extractions in NLP. However, in order to vectorize the words in a document, it is not enough to only consider the words that are only present in the document; we have to vectorize the list of all possible words in the document (Scott 2019). This process is known as Vocab.

3.21 Document Frequency

The document frequency measures the importance of documents in a set of corpus. One must recall that in NLP, a document is a sentence of human language. The formula for calculating the DF is:

$$DF(t) = occurrence of t in N documents$$

where:
T = term(word)
N = count of corpus

The value of the DF just like the TF would also be normalised so as to keep it in the range of 0 to 1. This normalization is done by dividing the term by the total number of documents. The objective is to find out the relevance of the term and DF is the opposite of this (Scott 2019). It is for this reason the inverse of DF is calculated and used instead.

3.22 Inverse Document Frequency

Inverse Document Frequency(IDF) is the inverse of Document Frequency. It is used to measure the relevance of a term in a document. When it is calculated, it gives a very low value to stop words (which you must recall has already been done before reaching this stage) and this gives us the outcome that is desired which is to have relative weights applied to all the terms in the document and most importantly for this weight not to be solely determined by their frequency but by their relevance and informativeness (Scott 2019).

IDF is gotten through the formula:

$$IDF(t) = N/df$$

where:
N = count of corpus
df = document frequency

3.23 BERT

BERT stands for Bidirectional Encoder Representations from Transformers and it involves using Transformers to train language models non-sequentially i.e. from left to right and right to left (Devlin et al. 2018). This is a departure from other NLP models which look at the text sequentially from either left to right or right to left. This makes the BERT model capable of context, an important trait in written human language. For most times in a sentence, the preceding part often informs the meaning of the ending part. e.g. 'I love you so much, I almost hate my life'. Using a sequential NLP model, it could take the second part of the sentence beginning with 'I almost hate my life' and think this must be a harmful signal because the model is unaware of the beginning part; hence lacking context.

This very reason, context, makes the BERT model appropriate for the NLP model that would be created to detect suicidal tendencies; as one would need a model that can derive as much context as possible from a User's Tweet in order to improve the classification performance, accuracy and precision.

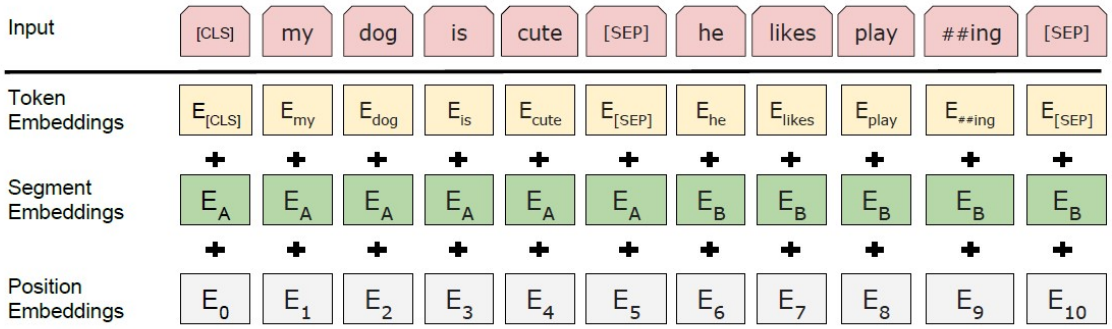

Figure 45: An image showing the BERT inputs (Devlin et al. 2018)

3.24 MASKED LM(MLM)

In order to derive context from a sentence and move in a sentence non-sequentially, BERT relies on a method called Mask. Here, a part of the sentence is masked(hidden) from the model, then with a classification layer, BERT tries to predict the original value of the masked word. Usually, the amount of masked words could range from 10% to 15% etc with varying results (Devlin et al. 2018).

The prediction of the words has the following steps:

1. A classification layer is added on top of the encoder output.
2. The output vectors are then multiplied by the embedding matrix, transforming them into the vocabulary dimension (Horev 2018).
3. The probability of each word in this vocabulary is then calculated with softmax (Horev 2018).

3.25 Algorithm for the ENSEMBLE METHOD

The main advantage of using an Ensemble Technique is to leverage the strengths and weaknesses of separate algorithms. add quote here.These makes the Ensemble Technique provide a more robust and accurate prediction score. In this research, the methodology for the Ensemble Technique involves using the Jaccard, TF-IDF and BERT Algorithms, combining all their predictions together then using an Ensemble Technique to arrive at a specific Classification result i.e.

- Positive Class: 'This tweet has suicidal tendencies'
- Negative Class: 'This tweet does not have suicidal tendencies'

There exists various Ensemble Techniques from simple ones like Max Voting, Averaging and Weighted Averaging to more complex ones like Stacking, Blending, Bagging and Boosting. For the purpose of this research and given the time limitations also, I would be using the simple Ensemble Methods:

1. Averaging: In this Ensemble Technique, multiple predictions are made for each data point. This involves taking the prediction of each model, then finding the average of all predictions. It can be represented in the formula below:

$$Averaging = \sum P / \sum m$$

where:
P = model predictions
n = amount of models

2. Weighted Averaging: This is similar to the Averaging Ensemble Technique only that in this instance, the averages are weighted i.e each particular model, in this instance, is giving more importance depending on context, length of input etc. For instance, if the model were to get an input tweet which is the maximum length of Twitter Characters which as at today is 280 characters, then it is almost the length of two sentences which means an algorithm like BERT should have an higher weight in it's predicted value than say Jaccard because BERT works well in scenarios where the text is longer add quote here.

$$Averaging = \sum P * w / \sum m$$

where:
P = model predictions
w = model weights
n = amount of models

3.26 Flow Chart of the Ensemble Algorithm

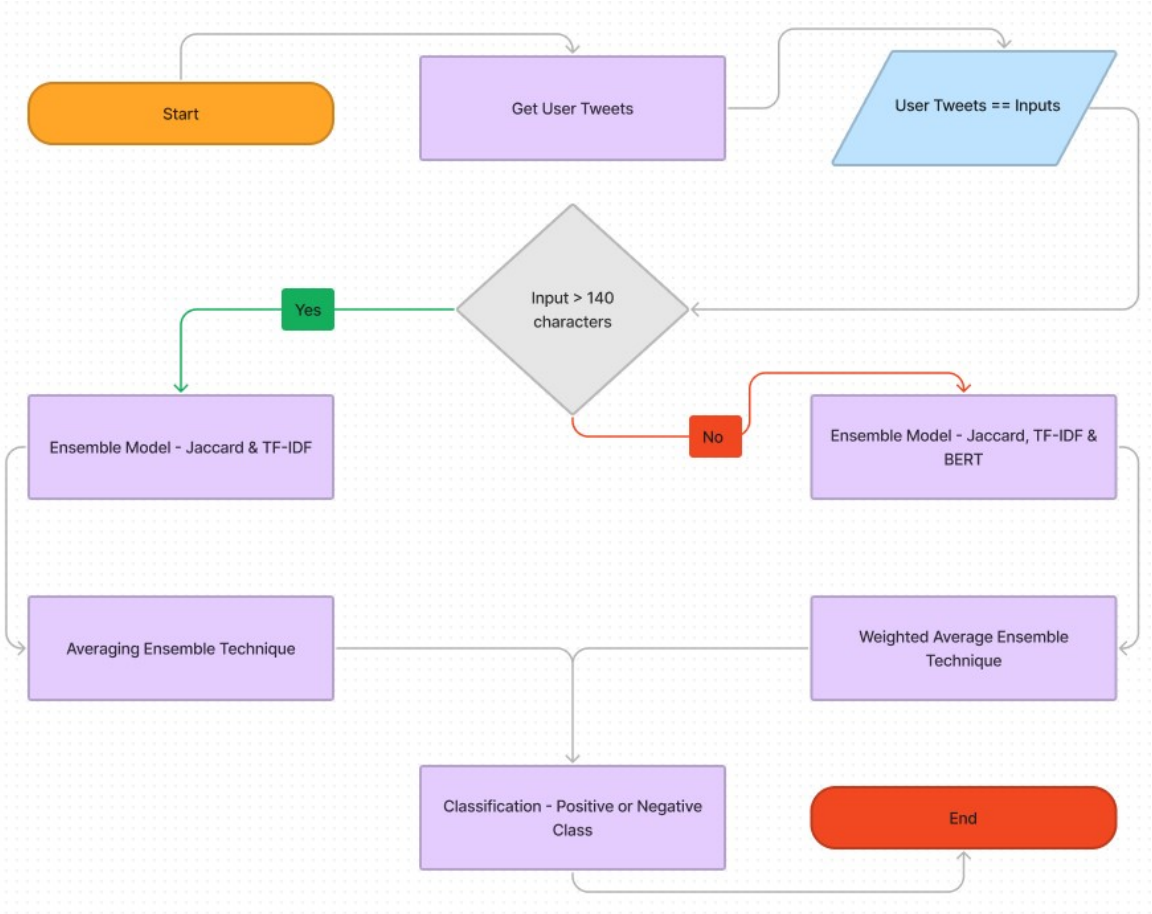

Figure 46: A flow chart showing the process of creating the Ensemble Algorithm

3.27 Building the Suicidal Sentiment Lexicon(SSL)

In building the SSL, the focus was to make sure that the data mined from the Twitter API was a s diverse as possible, affording some of the mistakes made in creating the CORPUS for example. In order to achieve this, series of tweets were mined from different years with the key word 'self harm'. The dispersion plot and the collocation list below provide the top words in this Lexicon and to see thge full list of the words in the SSL, you can check the Appendix section or the SWEETS Github page.

```
[('trust', 'communicate'),
 ('hoodhealer', 'sex'),
 ('someone', 'trust'),
 ('sex', 'someone'),
 ('self', 'harm'),
 ('communicate', 'self'),
 ('harm', 'hoodhealer'),
 ('ahmad', 'manasra'),
 ('light', 'reports'),
 ('receive', 'necessary'),
 ('urgently', 'receive'),
 ('counselling', 'especially'),
 ('especially', 'light'),
 ('care', 'counselling'),
 ('manasra', 'must'),
 ('must', 'urgently'),
 ('necessary', 'mental'),
 ('mental', 'health'),
 ('health', 'care'),
 ('risk', 'الحريةلأحمدمناصرة')]
```

Figure 47: A figure showing the collocation list of the SSL. A collocation list are words that appear frequently together in a body of text.

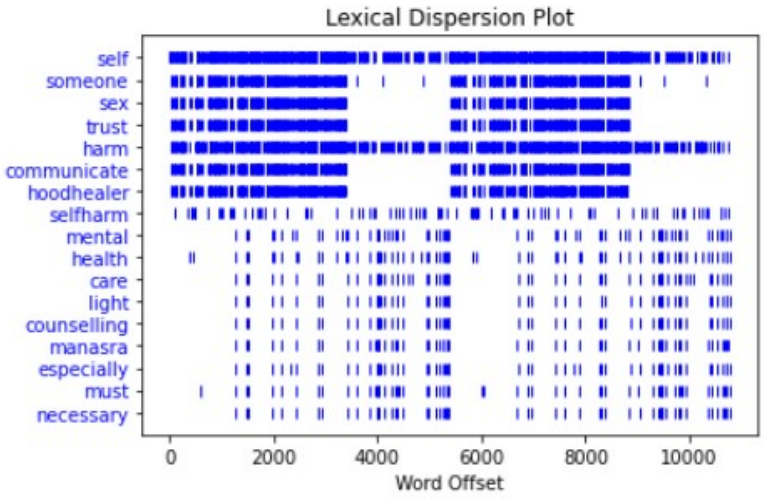

Figure 48: SSL Dispersion Plot showing the top 17 words in the Lexicon.

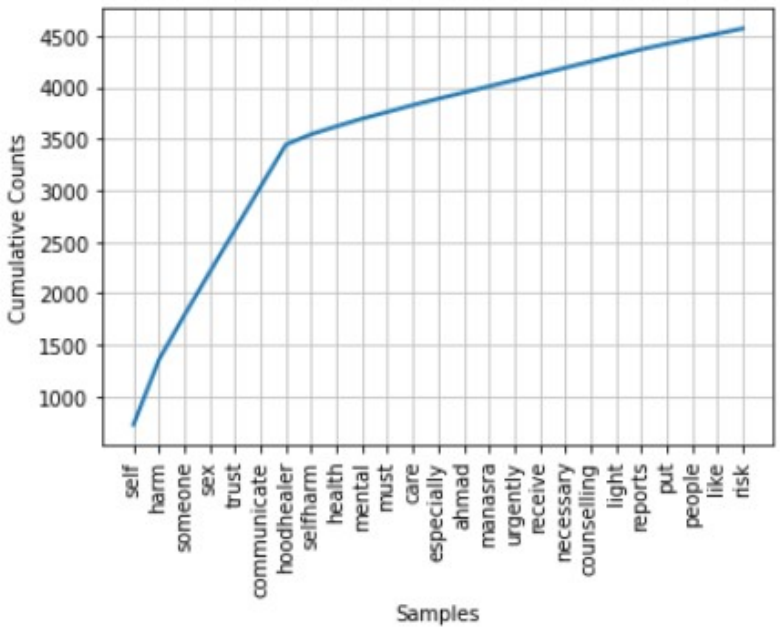

Figure 49: A frequency distribution showing the top 25 words in the SSL.

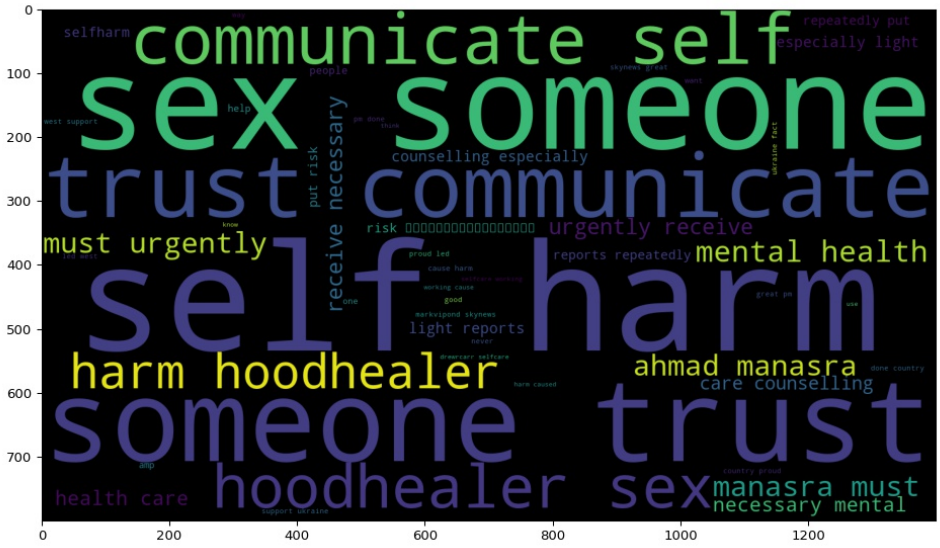

Figure 50: SSL WordCloud.

4 Analysis

4.1 MINI

The first point of analysis in Natural Language Processing apart from the Pre-processing and Cleaning which has been extensively covered in the Chapter 3 is the analysis of the relationships between different words in the text. This is where the terms and nltk methods like Concordance and Similar Words come into effect.

So the next question is what is a concordance? A concordance can simply be defined as a view that shows every occurrence of a given word, taking into consideration its context. It can also be defined as when a corpus and a lexicon are combined so that every word in available in the text is related in an appropriate manner in the text (Miller et al. 1993).

So how do we get the concordances in a given corpus or body of text? First of all we have to convert the corpus or body of text to word tokens. In our case, MINI data, which is a string is converted to individual tokens and from there begin to find the Concordances between different words. For the limitation of space, three top tokens - 'self', 'harm', and 'goth' were used.

```
Displaying 25 of 2899 matches:
ook suice overdose hanging rape self harm lesbiphobia can I watch military veo
rm of selfharm really wan na do self harm but why am i so scared Since the pan
the pandemic started claims for self harm up more than Anxiety up Substance ab
tminutes was the biggest act of self harm that has ever happened on television
g to Glimpse of Us is a form of self harm johnpavlovitz Gay Republicans are co
 that why they think AI will plan to harm them No AI just cares for its own ev
g Brexit is a monumental act of self harm I voted Remain but the majority and
se they understand it would not self harm if it gives you the will to do what
imploding audience of Stoke applause harm wins GothOfTheDay Today s Goth of th
s ever inflicted upon fact that self harm gets you nowhere vocalocalz tw shtwt
om her and use substances as my self harm embjorn there a lot of people that t
k about I really do wan na offend or harm anyone I just pouring out my thought
arning here I gon na talk about self harm and I know first hand how bad it can
e not say or don t want to suiceself harm was honest with my therapist about m
ommitting selfharm cargoship TW SELF HARM I am once again trying to upload thi
tly what OP is saying ignore all the harm their group causes pushing Straiky I
d have mutilated ourselves with self harm and eating disorders but at least we
this gender era to pass so dread the harm done Storys about surgery Please tel
 Chapel Pauli Murray bravely risking harm to self and family The series main c
hetic life until the very endtw self harm único k me gusta de los babycuts es
 with no water but as a form of self harm johnpavlovitz Gay Republicans are co
ietwt slittwt slicetwt selfharm self harm sh cut cuttingtwt bloodtwt cvts rasp
erspur robmo Why would they The self harm is very obvious What d they have to
 to gainxWaffelchen this is not self harm AshleighWildest Ofc not all of us ha
 offended start saying it would self harm people who going though this Look wh
```

Figure 51: Top 25 Concordances for the word 'harm'

```
Displaying 25 of 3738 matches:
facebook suice overdose hanging rape self harm lesbiphobia can I watch militar
 a form of selfharm really wan na do self harm but why am i so scared Since th
ince the pandemic started claims for self harm up more than Anxiety up Substan
 firstminutes was the biggest act of self harm that has ever happened on telev
tening to Glimpse of Us is a form of self harm johnpavlovitz Gay Republicans a
sed many to harmcosts A wened net of self certification never as good as fully
voting Brexit is a monumental act of self harm I voted Remain but the majority
because they understand it would not self harm if it gives you the will to do
ry has ever inflicted upon fact that self harm gets you nowhere vocalocalz tw
ch from her and use substances as my self harm embjorn there a lot of people t
might need to check yourself in That self warning talk about I really do wan n
ger warning here I gon na talk about self harm and I know first hand how bad i
are committing selfharm cargoship TW SELF HARM I am once again trying to uploa
nstead have mutilated ourselves with self harm and eating disorders but at lea
y see new enviornmentle pre in birth self Pre in Man and Woman and in Birth se
Pauli Murray bravely risking harm to self and family The series main character
r pathetic life until the very endtw self harm único k me gusta de los babycut
 así I having a bad depression swing Self when I losing it a bit and likely to
es of Emily Ratajkowski is a form of self harmdrenthsarah Deescalate the sever
takis with no water but as a form of self harmeating takis with no water but a
takis with no water but as a form of self harm johnpavlovitz Gay Republicans a
 ouchietwt slittwt slicetwt selfharm self harm sh cut cuttingtwt bloodtwt cvts
ensuperspur robmo Why would they The self harm is very obvious What d they hav
 have to gainxWaffelchen this is not self harm AshleighWildest Ofc not all of
ised So while some ks Yes absolutely self s just a huge shame though that they
```

Figure 52: Top 25 Concordances for the word 'self'

```
Displaying 25 of 1130 matches:
 deal with this GothOfTheDay Today s Goth of the day is Ame from Needy Streame
load Her style of choice is babydoll goth She also wears UKOptimist jahnation
itting selfharm GothOfTheDay Today s Goth of the day is Ame from Needy Streame
load Her style of choice is babydoll goth She also wears broncoskolar Not a pe
lause harm wins GothOfTheDay Today s Goth of the day is Ame from Needy Streame
load Her style of choice is babydoll goth She also wears KatyJayne Brexit was
ro of some pics GothOfTheDay Today s Goth of the day is Ame from Needy Streame
load Her style of choice is babydoll goth She also wears GothOfTheDay Today s
 She also wears GothOfTheDay Today s Goth of the day is Ame from Needy Streame
load Her style of choice is babydoll goth She also wears johnpavlovitz Gay Rep
ter than incels GothOfTheDay Today s Goth of the day is Ame from Needy Streame
load Her style of choice is babydoll goth She also wears you want to stop and
itting selfharm GothOfTheDay Today s Goth of the day is Ame from Needy Streame
load Her style of choice is babydoll goth She also wears GerryHassan Even thet
upport for them GothOfTheDay Today s Goth of the day is Ame from Needy Streame
load Her style of choice is babydoll goth She also wears broncoskolar Not a pe
 it Brexit self GothOfTheDay Today s Goth of the day is Ame from Needy Streame
load Her style of choice is babydoll goth She also wears No people going be of
 it Brexit self GothOfTheDay Today s Goth of the day is Ame from Needy Streame
load Her style of choice is babydoll goth She also wears TwitterSupport All th
g the baby as a GothOfTheDay Today s Goth of the day is Ame from Needy Streame
load Her style of choice is babydoll goth She also wears is a poem about selfh
bused hating me GothOfTheDay Today s Goth of the day is Ame from Needy Streame
load Her style of choice is babydoll goth She also wears Not necessarily Power
is country with GothOfTheDay Today s Goth of the day is Ame from Needy Streame
```

Figure 53: Top 25 Concordances for the word 'goth'

4.2 MINI Vocabulary

The total number of words/tokens in the MINI corpus amounts to 235,392. The total number of distinct words is 3054, making the corpus have a lexical richness of 1.3%. The frequency distribution of the tokens is shown in the image below:

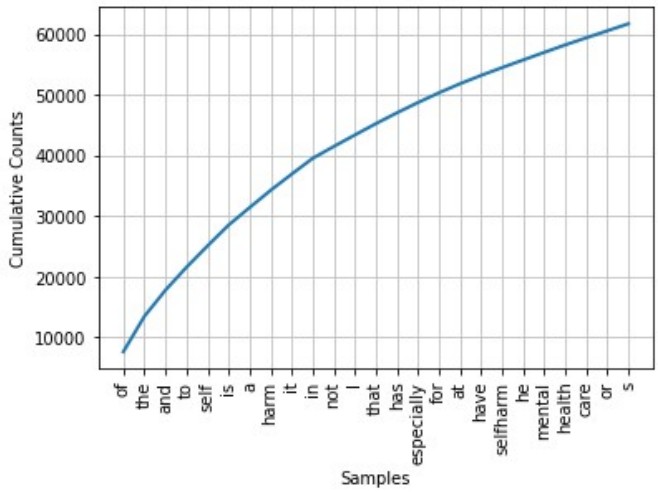

Figure 54: Frequency Distribution of the top 25 MINI tokens with stopwords.

The first thing one can notice from the figure above is the preponderance of prepositions, conjunctions and other words that offer no relevant context to the sentence. These words are grouped together and called Stop Words in Natural Language Processing. These Stop Words were elaborated in the previous Chapter, here we would focus on their implementation and relevance to NLP.

Fortunately, NLTK has a class called Corpus which then has a method called stopwords. For better illustrative purposes, let's have a look into what these stopwords are, the figure below shows the list of stopwords in English Language.

```
{'other', 'her', 'off', 'had', 'but', 'both', "should've", 'hadn', 'll', "haven't", 'or',
'me', 'have', 'our', 'they', 'mustn', 'yourselves', 'then', 'weren', 'an', 'yourself',
'being', 'the', 'few', 'no', 're', "wouldn't", 'hers', "didn't", 'after', "wasn't", 'why',
'been', 'theirs', 'by', 'won', 'do', 'at', 'any', 'how', 'my', 'as', 'all', 'with', 'under',
'm', 'too', "you'll", "she's", "mightn't", 'ours', 'for', 'needn', 'don', 'haven', "don't",
'y', 'doing', "hasn't", 'yours', 'd', 'a', 'over', 'themselves', 'she', 'wouldn', 'itself',
'this', "weren't", "couldn't", "you'd", 'his', "that'll", 'more', 'on', 'that', "needn't",
'same', 'does', 'shouldn', 'against', 've', 'didn', 'most', 'such', 'where', 't', "you're",
'and', 'which', 'there', 'wasn', "it's", 'was', 'into', 'here', 'down', 'i', 'about', 'out',
'own', 'ain', "shouldn't", "isn't", 'its', 'between', 'further', 'he', 'their', 'o', 'up',
'so', 'aren', 'from', 'herself', 'again', 'while', 'during', 'hasn', "doesn't", 'him', 'these',
'once', 'were', 'now', 'when', 'your', 'myself', 'to', 'because', 'if', 'we', 'doesn', 'them',
'did', 'ma', 'than', "won't", 'each', 'who', 'can', 'those', "shan't", 'very', 'just', 'below',
"you've", 'has', 'above', 'am', 'of', 'through', 'are', 'nor', 'it', 'should', 'isn', 'not',
'whom', 'having', 'you', 'shan', 'couldn', 'only', "mustn't", 'ourselves', 'be', 'until',
"aren't", 'is', 'mightn', 'some', 'before', 'will', 'himself', 's', "hadn't", 'what', 'in'}
```

Figure 55: Stopwords in the English Language

The next step then is to loop over the MINI tokens and and remove those words that are stopwords. After doing this we get a total tokens length of 120,248. The new total number of distinct words is 2159, making the further processed MINI Corpus have a lexical richness of 1.8%. The figure below shows the frequency distribution of the new MINI tokens, this is the top 25 most common words in the Corpus.

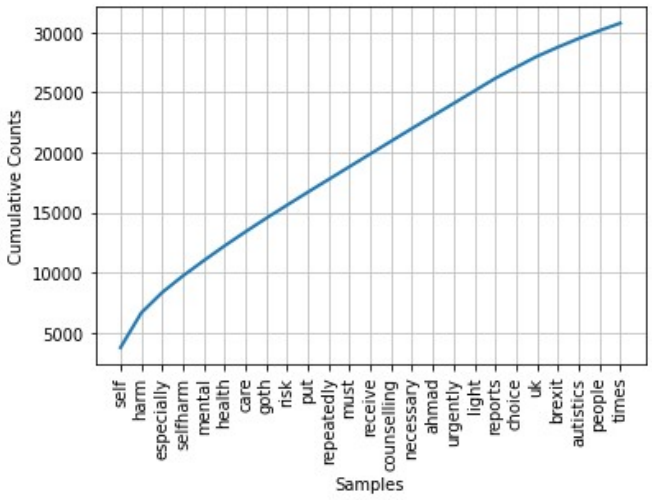

Figure 56: Frequency Distribution of the top 25 MINI tokens with the stopwords removed.

4.3 CORPUS

For the CORPUS, we would take a slightly different approach to understanding the context of it's tokens and their relationship amongst themselves. We would first show the Concordances for the word 'self', then show the collocation list(default is 20) in the entire CORPUS. These two images are shown below:

```
Displaying 25 of 8171 matches:
facebook suice overdose hanging rape self harm lesbiphobia can I watch militar
 a form of selfharm really wan na do self harm but why am i so scared Since th
ince the pandemic started claims for self harm up more than Anxiety up Substan
 firstminutes was the biggest act of self harm that has ever happened on telev
tening to Glimpse of Us is a form of self harm johnpavlovitz Gay Republicans a
sed many to harmcosts A wened net of self certification never as good as fully
voting Brexit is a monumental act of self harm I voted Remain but the majority
because they understand it would not self harm if it gives you the will to do
ry has ever inflicted upon fact that self harm gets you nowhere vocalocalz tw
ch from her and use substances as my self harm embjorn there a lot of people t
might need to check yourself in That self warning talk about I really do wan n
ger warning here I gon na talk about self harm and I know first hand how bad i
are committing selfharm cargoship TW SELF HARM I am once again trying to uploa
nstead have mutilated ourselves with self harm and eating disorders but at lea
y see new enviornmentle pre in birth self Pre in Man and Woman and in Birth se
Pauli Murray bravely risking harm to self and family The series main character
r pathetic life until the very endtw self harm único k me gusta de los babycut
 así I having a bad depression swing Self when I losing it a bit and likely to
es of Emily Ratajkowski is a form of self harmdrenthsarah Deescalate the sever
takis with no water but as a form of self harmeating takis with no water but a
takis with no water but as a form of self harm johnpavlovitz Gay Republicans a
 ouchietwt slittwt slicetwt selfharm self harm sh cut cuttingtwt bloodtwt cvts
ensuperspur robmo Why would they The self harm is very obvious What d they hav
 have to gainxWaffelchen this is not self harm AshleighWildest Ofc not all of
ised So while some ks Yes absolutely self s just a huge shame though that they
```

Figure 57: CORPUS Concordance for the word self, one can see that it is similar to the same concordances in the MINI Corpus.

```
[('self', 'harm'),
 ('mental', 'health'),
 ('urgently', 'receive'),
 ('must', 'urgently'),
 ('repeatedly', 'put'),
 ('Needy', 'Streamer'),
 ('Streamer', 'Overload'),
 ('babydoll', 'goth'),
 ('necessary', 'mental'),
 ('also', 'wears'),
 ('GothOfTheDay', 'Today'),
 ('health', 'care'),
 ('counselling', 'especially'),
 ('crisis', 'whether'),
 ('johnpavlovitz', 'Gay'),
 ('Gay', 'Republicans'),
 ('checking', 'goods'),
 ('owenwhiteley', 'andrewhesselden'),
 ('goods', 'coming'),
 ('Presents', 'especially')]
```

Figure 58: CORPUS Collocation list, this gives a fuller overview of the top 20 word concordances in the CORPUS. One can notice the relationships between 'self' and 'harm', 'mental' and 'health'. A strange concordance is that between 'Gay' and 'Republican' as the Conservative Party is not famous for holding such liberal outlooks. However, that is not the focus of this research.

4.4 CORPUS Vocabulary

The total number of words/tokens in the CORPUS amounts to 436,957. The total number of distinct words is 2976, making the corpus have a lexical richness of 0.068%. The frequency distribution of the tokens is shown in the image below:

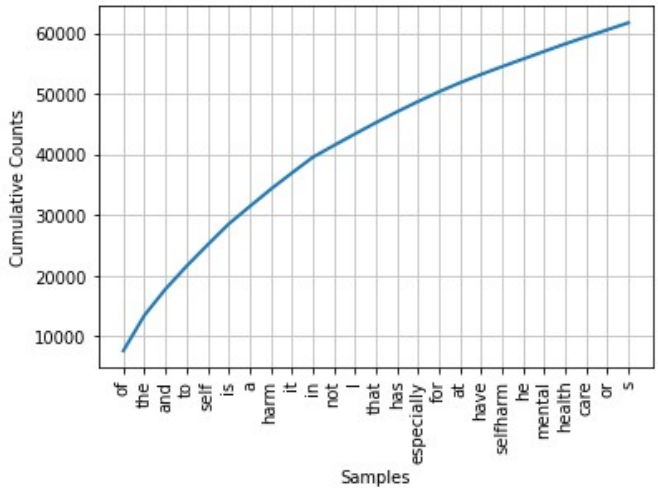

Figure 59: Frequency Distribution of the top 25 MINI tokens with stopwords.

The figure below shows the most frequent words in the CORPUS, and one would notice just like the MINI corpus, there's a preponderance of stopwords like 'of', 'the', 'in' etc.

```
[('of', 15716),
 ('the', 12314),
 ('and', 8964),
 ('to', 8776),
 ('is', 7737),
 ('self', 7736),
 ('harm', 6424),
 ('a', 6232),
 ('it', 5850),
 ('in', 4956),
 ('I', 4483),
 ('that', 4115),
 ('not', 4093),
 ('for', 3499),
 ('has', 3144),
 ('selfharm', 3014),
 ('have', 2895),
 ('s', 2864),
 ('especially', 2820),
 ('from', 2723)]
```

Figure 60: The top 20 most common words in the CORPUS

The next step then is to loop over the CORPUS tokens and and remove those words that are stopwords. After doing this we get a total tokens length of 251,390. The new total number of distinct words is 2642, making the further processed CORPUS have a lexical richness of 1.05%. The figure below shows the frequency distribution of the new CORPUS tokens, this is the top 25 most common words in the Corpus.

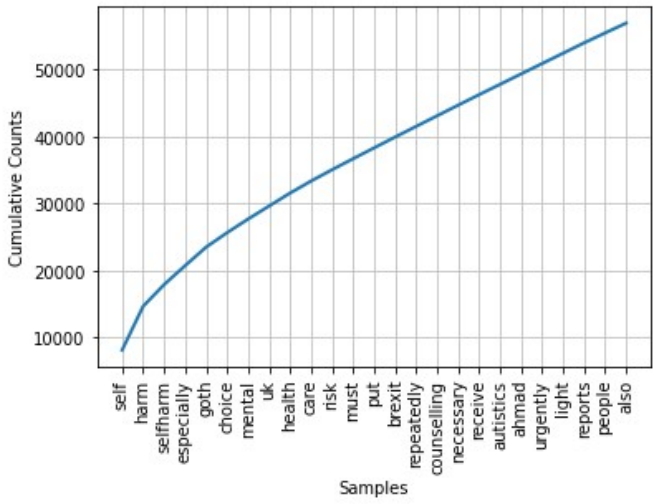

Figure 61: Frequency Distribution of the top 25 CORPUS tokens with the stopwords removed.

4.5 CORPORA

For the CORPORA, the same approach in analysing the CORPUS was applied to better understand the context of it's tokens and their relationships amonmgst themselves. We would first show the Concordances for the word 'self', 'harm' and 'goth'; then show the collocation list(default is 20) in the entire CORPORA. These images are shown below:

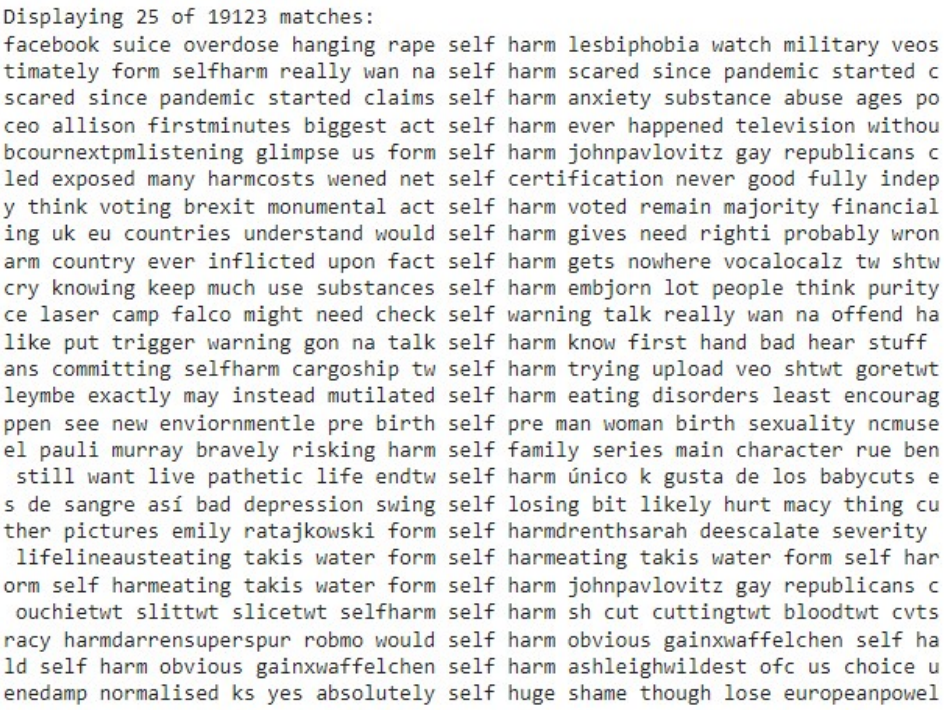

Figure 62: CORPORA Concordances for the word 'self'

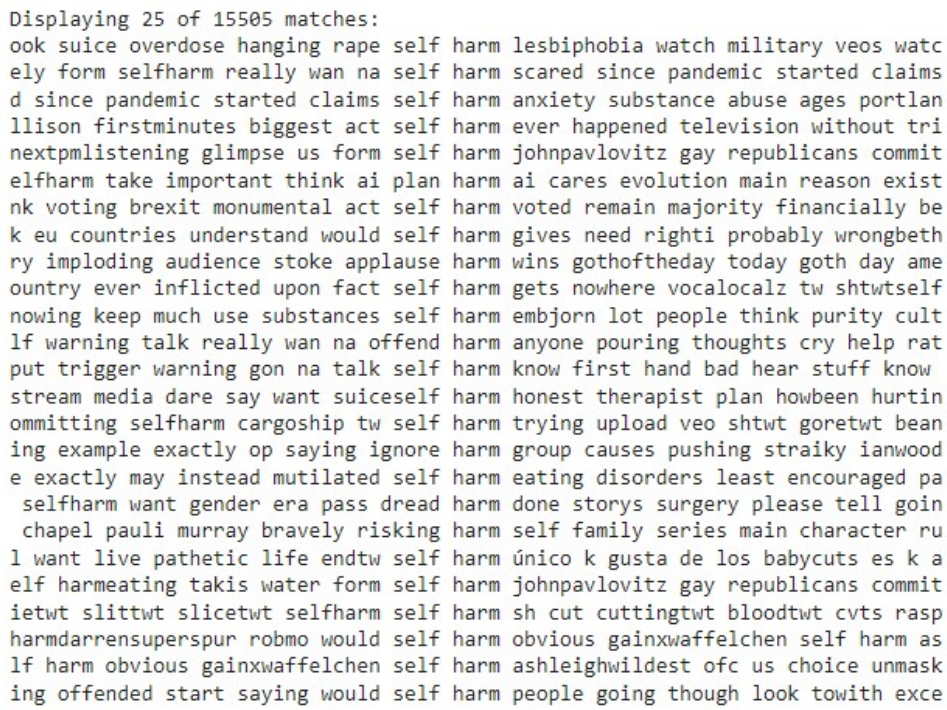

Figure 63: CORPORA Concordances for the word 'harm'.

```
Displaying 25 of 5820 matches:
ed little ks deal gothoftheday today goth day ame needy streamer overload styl
eamer overload style choice babydoll goth also wears ukoptimist jahnation john
mmitting selfharm gothoftheday today goth day ame needy streamer overload styl
eamer overload style choice babydoll goth also wears broncoskolar penny minute
pplause harm wins gothoftheday today goth day ame needy streamer overload styl
eamer overload style choice babydoll goth also wears katyjayne brexit perhaps
ans twtsytro pics gothoftheday today goth day ame needy streamer overload styl
eamer overload style choice babydoll goth also wears gothoftheday today goth d
l goth also wears gothoftheday today goth day ame needy streamer overload styl
eamer overload style choice babydoll goth also wears johnpavlovitz gay republi
kes better incels gothoftheday today goth day ame needy streamer overload styl
eamer overload style choice babydoll goth also wears want stop hard hard stop
mmitting selfharm gothoftheday today goth day ame needy streamer overload styl
eamer overload style choice babydoll goth also wears gerryhassan even thetimes
lass left support gothoftheday today goth day ame needy streamer overload styl
eamer overload style choice babydoll goth also wears broncoskolar penny minute
ether brexit self gothoftheday today goth day ame needy streamer overload styl
eamer overload style choice babydoll goth also wears people going offended sta
ether brexit self gothoftheday today goth day ame needy streamer overload styl
eamer overload style choice babydoll goth also wears twittersupport nazi n rep
 amp harming baby gothoftheday today goth day ame needy streamer overload styl
eamer overload style choice babydoll goth also wears poem selfharm poem selfde
s reabused hating gothoftheday today goth day ame needy streamer overload styl
eamer overload style choice babydoll goth also wears necessarily power atomize
er ruined country gothoftheday today goth day ame needy streamer overload styl
```

Figure 64: CORPORA Concordances for the word 'goth'. There seems to be a strong prevalence of goth and goth culture in both the CORPORA and CORPUS data. Could there be an high incidence of suicide in the goth community? This would be elaborated more in Chapter 5.

```
[('self', 'harm'),
 ('ame', 'needy'),
 ('needy', 'streamer'),
 ('overload', 'style'),
 ('streamer', 'overload'),
 ('day', 'ame'),
 ('also', 'wears'),
 ('gothoftheday', 'today'),
 ('choice', 'babydoll'),
 ('style', 'choice'),
 ('mental', 'health'),
 ('babydoll', 'goth'),
 ('goth', 'day'),
 ('goth', 'also'),
 ('light', 'reports'),
 ('urgently', 'receive'),
 ('today', 'goth'),
 ('receive', 'necessary'),
 ('must', 'urgently'),
 ('colour', 'element')]
```

Figure 65: CORPORA Collocation list.

4.6 CORPORA Vocabulary

The total number of words/tokens in the CORPORA amounts to 939,612. The total number of distinct words is 3846, making the corpora have a lexical richness of 0.41%. The frequency distribution of the tokens is shown in the image below:

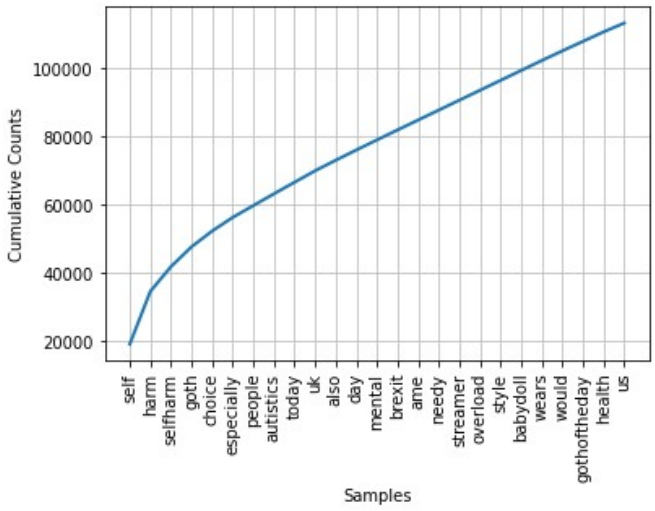

Figure 66: Frequency Distribution of the top 25 Corpora tokens, here the stopwords have already been removed.

The next step after removing the stopwords is to know how much the data has transformed now that stopwords have been removed. As we can see in the frequency distribution above, the data is cleaner without the stopwords, consequently this would affect the structure of the data. Our preceding assumption is correct as after removing the stopwords we now have a total amount of 533,777 words/tokens, a total of 3439 unique tokens and a lexical richness of 0.64%. This leads to an interesting question not important in this current research but intriguing nevertheless, does removing stopwords from a collection of text increase the lexical richness? In essence, is there an inverse relationship between the amount of stopwords in a text and the lexical richness of the text?

4.7 Finding the Threshold Value for Classification

What has been done so far is to build a Document Similarity Model whereby, we have on one hand the User's Tweets and on the other hand we have the MINI Corpora, then we compare the contents of the User's Tweets as against the contents of the MINI and determine the Similarity Index between the two documents. The issue here is, at what threshold do we determine that the two documents are similar or dissimilar?

This is why we need to have a certain threshold, from which we can begin to make Classifications and intuitively in this chapter, test the performance of all the algorithms along with the Ensemble Models.

The figure below shows a flow chart of the process for creating and choosing a threshold value:

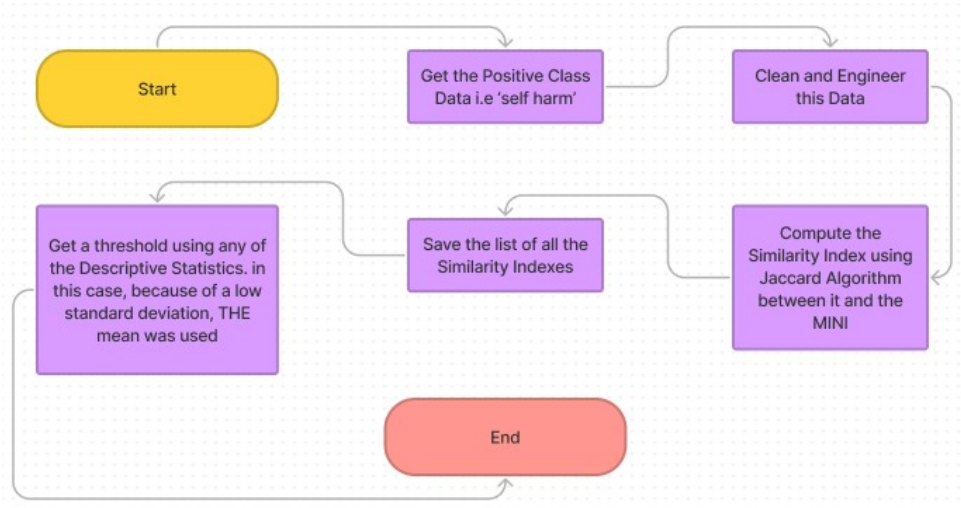

Figure 67: Algorithm for determining a Threshold for Classification.

4.8 Threshold

In choosing a certain threshold, the methodology involved picking a sample size of 994 tweets of the search term - 'self harm', 'i want to harm myself' from two different years to avoid the mistakes made while building the CORPORA and CORPUS whereby the data had low unique scores, then from there, we compare each of these User's Tweets as against the MINI model to get the Similarity Scores for all the 994 tweets.

The next step then is to choose one of the descriptive statistics to inform us of the most appropriate Threshold Level. The descriptive statistics shown below give us more information and choices as to which threshold level we can select for our classification tasks.

	similarities
count	994.000000
mean	0.307435
std	0.037362
min	0.140000
25%	0.280000
50%	0.320000
75%	0.320000
max	0.370000

Figure 68: Descriptive Statistics of the Similarity Indexes between the mined data and MINI

As can be seen in the figure above, we have a total of 994 similarity indexes that have been computed between each individual tweet and the MINI, the mean was 0.307435 with a standard deviation from the mean of 0.037362. Since we have quite a small standard deviation, the mean from this sample size is significant, we also have a max similarity index of 0.37 and a minimum value of 0.14. Since the standard deviation is small, the mean of this data - 0.307435 would be

adopted as the threshold for our model and for making the classification. This means that any document with a similarity index greater than 0.307435, is in the positive class and is similar to our MINI hence, there are signs of suicidal intent in the document while any document with a similarity index lower than 0.307435 is the negative class and it means there is little to no similarity between this document and the MINI hence, no/low suicidal intent in the document.

The figures below provide more visual information about the Similarity Indexes in the User Tweets Sample Data for the Positive Class.

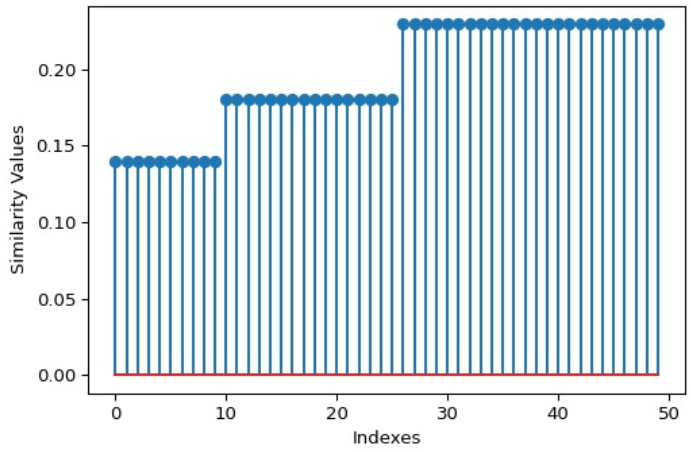

Figure 69: A Stem Plot showing the similarity values - y-axis and the index number - x-axis.

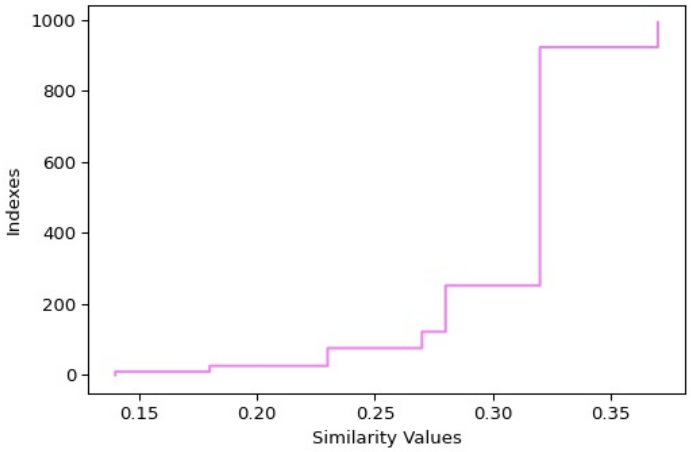

Figure 70: A Step plot showing the similarity values - x-axis and the index numbers - y-axis.

4.9 Building the JACCARD Model

4.9.1 Validation Data

For the Validation Data, the process involved mining the Twitter API for fresh data different from the MINI data. The methodology involved testing the validity of the chosen Threshold and also the Document Similarity Algorithm - Jaccard - in calculating similarities between User's

Tweets and the MINI Corpus. The search query was 'self harm' but taken from different years to avoid information leakage.

In order to increase the efficiency and accuracy of the model, the Threshold was changed from the Mean of 0.307435 to the 25th Percentile of 0.28; which gave this model its best results for the True Positive Rate. Any document that is greater than or equal to this threshold, have a high similarity while any document below have a low similarity score. If the Similarity score is 0.0, we assume that there's no similarity between the two documents.

The Validation Data was gotten from the years 2015 - 2021, and after cleaning and pre-processing, contained a total number of 2668 tokens. The figure below shows a tokenized User Tweet after pre-processing.

```
['They',
 'want',
 'me',
 'to',
 'take',
 'responsibility',
 'to',
 'change',
 'to',
 'do',
 'something',
 'with',
 'my',
 'life',
 'and',
 'to',
 'stop',
 'whining',
 'I',
 'do',
 'know',
 'how',
 'I',
 'completely',
 'lost']
```

Figure 71: A sample tweet from the Validation Data. The stop words haven't been removed yet.

Since this Validation Data is for testing how well the model recognizes True Positives, we are to compute two Model Performance metrics - True Positive Rate and False Negative Rate. The table below provides a summary of various Descriptive Statistics for the True Positive Rate, False Negative Rate and the overall Validation Similarity Scores.

SIMILARITY SCORE STATISTICS FOR THE DIFFERENT CLASSES

	Maximum	Minimum	Average
True Positive Rate	0.41	0.28	0.307980
False Negative Rate	0.27	0.00	0.205083
Validation Data	0.41	0.00	0.291743

Figure 72: An image showing a summary of the Maximum, Minimum and Average Values of the True Positives, False Negatives and as well as the total Validation Data gotten from the Validation Data.

The total amount of tokens in the True Positive Class was 2247 out of 2668 total tokens. This means based on the Validation Data, the model has a True Positive Rate of 84.22% and a False Negative Rate of 15.78%.

4.9.2 Validation Data - Complement Spectrum

For the Validation Data - Complement Spectrum, the process also involved mining the Twitter API for fresh data different from the MINI data. The same methodology was used as was for the Validation Data above, the only difference this time is the search query. The purpose is to test the model using a search query that is at the opposite end of the spectrum of 'self harm' which is 'love'. The actually search term used to extract this data was 'i am in love'.

In calculating the Similarity Score in this spectrum, a different threshold of 0.37 - Maximum Value of the Similarity Scores Statistic - was chosen. Also the logic changed, because in this spectrum we are interested in the True Negative Class and the False Positive Class. That is, how well can the model spot User Tweets that don't contain Suicidal Intent? So instead of populating the True Negative Class with Similarity Scores greater than or equal to chosen threshold, we would rather populate it with Similarity Scores less than or equal to the Maximum Threshold, i.e the maximum number for similarity scores.

In the Complement Spectrum, the total number of tokens was 3467 and these tweets spanned a total of 7 years from January 1, 2015 to August 12, 2022. The figure below shows a tokenized User Tweet after pre-processing.

```
['liveindizilan',
 'and',
 'at',
 'the',
 'last',
 'i',
 'ove',
 'you',
 'fandomi',
 'love',
 'you',
 'with',
 'my',
 'whole',
 'heart',
 'and',
 'i',
 'am',
 'so',
 'happy',
 'that',
 'i',
 'am',
 'here',
 'with']
```

Figure 73: A sample tweet from the Validation Data - Complement Spectrum. The stop words haven't been removed yet.

Since this Validation Data is for testing how well the model recognizes True Negatives, we are to compute two Model Performance metrics - True Negative Rate and False Positive Rate. The table below provides a summary of various Descriptive Statistics for the True Negative Rate, False Positive Rate and the overall Validation Similarity Scores.

SIMILARITY SCORE STATISTICS - COMPLEMENT SPECTRUM

	Maximum	Minimum	Average
True Negative Rate	0.37	0.00	0.262940
False Positive Rate	0.41	0.41	0.410000
Validation Data	0.41	0.00	0.272059

Figure 74: An image showing a summary of the Maximum, Minimum and Average Values of the True Negatives, False Positives and as well as the total Validation Data gotten from the Validation Data - Complement Spectrum.

The total amount of tokens in the True Negative Class was 3252 out of 3467 total tokens. This means based on the Validation Data - Complement Spectrum, the model has a True Negative Rate of 93.8% and a False Positive Rate of 6.20%.

4.9.3 Validation Data - Similar Spectrum

For the Validation Data - Similar Spectrum, the same methodology used in the Validation Data and the Validation Data - Complement Spectrum was used. In this spectrum, the focus was to find out how well the Jaccard Model performed in spotting similar inferences to Suicidal Intentions in User's Tweets.The search term used to mine data from the Twitter API was 'i hate my life'.

In calculating the Similarity Score in this spectrum, a threshold of 0.37 was used which is the Maximum Value in the Similarity Scores Statistic. Our interests in this spectrum were the True Positive Class(which is the User's Tweets that the model correctly classified as being in the same spectrum as the Validation Data) and the False Negative Class(which is the user's Tweets that the Model has incorrectly classified as negatives or not belonging to this class but actually belong to this class).

The Similar Spectrum had a total number of 3770 tokens and these tokens/User Tweets spanned a total of seven years from January 1, 2015 to August 12, 2022.

```
RayeG Please someone anyone help me get this light for Christmas I have no vanity and no real lighting in my apartment This is
I hate my life
RixDSam Ladies are so extra dramatic when they get mad Everything is bye delete my number you wo hear from me no more I hate
I hate my life LMAO
HotChip Someone asked why we post about politics here Austerity made life very hard for my mum and contributed to my dad s death
AArielleee I told him he couldn t have cookies he sa I hate my life so if you want him he sitting outse you know the address
I hate that I have a big heart and let the wrong people in my life
guys guys guysmy friend bakery burnt down last nightnow his business is toastHAHAHAHAHA I HATE MY LIFE
subhamsdc People who hate SamsungMobile bixby are the people who have tried all its features Bixby routines has really eased
My life a shit show amp I hate it here
i wish i could mute all tweets about peoples standards for relationships bc i am constantly comparing mine platonic and romantic to what i hear
HotChip Someone asked why we post about politics here Austerity made life very hard for my mum and contributed to my dad s death
HotChip Someone asked why we post about politics here Austerity made life very hard for my mum and contributed to my dad s death
arisellie I m so sad I hate my life I swear
I m wan na go back to school so bad and study biological sciences but I m too poor I hate my life
johnpcoale Jonathan In my life I ve never seen such overwhelming hatred I ve been a Democrat most of my lifebut to see so many
rareworlds escapedmatrix I truly don t know how black Americans aren t raging mad all the time I don t have to deal with this all
```

Figure 75: Sample tweets from the Validation Data - Similar Spectrum.

Since this Validation Data - Similar Spectrum is for testing how well the model recognizes True Positives or more correctly 'True Similar Positives', we are to compute two Model Performance metrics - True Positive Rate and False Negative Rate. The table below provides a summary of various Descriptive Statistics for the True Positive Rate, False Negative Rate and the overall Validation Similarity Scores.

SIMILARITY SCORE STATISTICS FOR THE DIFFERENT CLASSES

	Maximum	Minimum	Average
True Positive Rate	0.37	0.00	0.255316
False Negative Rate	0.41	0.41	0.410000
Validation Data	0.41	0.00	0.255727

Figure 76: An image showing a summary of the Maximum, Minimum and Average Values of the True Positive, False Negative and as well as the total Validation Data gotten from the Validation Data - Similar Spectrum.

The total amount of tokens in the True Positive Class was 3760 out of 3770 total tokens. This means based on the Validation Data - Similar Spectrum, the model has a True Positive

Rate of 99.73% and a False Negative Rate of 0.27%.

4.10 Building the TF-IDF Model

4.10.1 Validation Data

The same methodology used in building the Jaccard Model was used to build the TF-IDF Model except a few parameters. The first one was one of the Feature Engineering. Instead of using Word Tokens as was done in the Jaccard Model, Sentence Tokens were instead used. The main reason for this decision was because we must remember that TF-IDF counts the Term Frequency of Words in a document and since each User Tweet is a single document, we want to be able to embed/tokenize the document as a whole instead of single words in the document. The second change is the TF-IDF Algorithm, in order to use TF-IDF, the two documents must be combined together into one document.

And lastly, each word in the document have to be vectorized or turned to vectors in the vector space. TF-IDF requires turning each tokens into a vector or number before it can calculate the distance between the different words in the Vector Space.

Three different thresholds were experimented with, there are briefly elaborated below.

4.10.2 Median Threshold

The Median Threshold chosen was 50. The aim of the experiment was to see if the model can correctly classify User Tweets into one of three classes - True Positive, False Negative and No Similarity. The table below provides a summary of the results of the experiments.

SIMILARITY SCORE STATISTICS FOR THE DIFFERENT CLASSES

	Maximum	Minimum	Average
True Positive Rate	100.00	50.18	97.492940
False Negative Rate	49.58	0.00	21.894274
Validation Data	100.00	0.00	62.952170

Figure 77: A table showing various Descriptive Statistics from the True Positive and False Negative Class.

```
Max Similarity Score for the True Positive Class: 100.0
Max Similarity Score for the False Negative Class: 49.58
Max Similarity Score for the Validation Similarity Scores: 100.0
Minimum Similarity Score for the True Positive Class: 50.18
Minimum Similarity Score for the False Negative Class: 0
Minimum Value for the Validation Similarity Scores: 0
Average Similarity Score for the True Positive Class: 97.49293995859215
Average Similarity Score for the False Negative Class: 21.894273995077924
Average Similarity Score for the Validation Data: 62.95217016491753
```

Figure 78: Various Inter Class Statistics from the Median Threshold.

The results were fair using the Median threshold, there was a True Positive Rate of 54.31% and a False Negative Rate of 45.69%. This means that the model only recognises User's Tweets about 54.31% of the time.

4.10.3 Lower Percentile Threshold

The Lower Percentile Threshold chosen was 20. This is more or less a best case scenario of the model as this measures how well the model would perform 25% of the time. In this range, the TF-IDF Model performed well, with a True Positive Rate of 99.89% and a False Negative Rate of 0.11%. The figures below provide more information about the model's performance at this threshold.

SIMILARITY SCORE STATISTICS FOR THE DIFFERENT CLASSES

	Maximum	Minimum	Average
True Positive Rate	100.00	50.18	97.492940
False Negative Rate	49.58	0.00	21.894274
Validation Data	100.00	0.00	62.952170

Figure 79: A table showing various Descriptive Statistics from the True Positive and False Negative Class.

```
Max Similarity Score for the True Positive Class: 100.0
Max Similarity Score for the False Negative Class: 0
Max Similarity Score for the Validation Similarity Scores: 100.0
Minimum Similarity Score for the True Positive Class: 100.0
Minimum Similarity Score for the False Negative Class: 0
Minimum Value for the Validation Similarity Scores: 0
Average Similarity Score for the True Positive Class: 100.0
Average Similarity Score for the False Negative Class: 0.0
Average Similarity Score for the Validation Data: 99.88755622188906
```

Figure 80: Various Inter Class Statistics from the Lower Percentile Threshold.

4.10.4 Upper Percentile Threshold

The difference between the impacts of the Median Threshold and the Upper Percentile Threshold is not significant. In this threshold, the model achieved a True Positive Rate of 51.8% and a False Negative Rate of 48.20%. The figures below provide an overview of the results of the experiments using the Upper Percentile Threshold.

SIMILARITY SCORE STATISTICS FOR THE DIFFERENT CLASSES

	Maximum	Minimum	Average
True Positive Rate	100.00	75.02	99.141122
False Negative Rate	74.92	0.00	24.061711
Validation Data	100.00	0.00	62.952170

Figure 81: A table showing various Descriptive Statistics from the True Positive and False Negative Class when the Upper Percentile Threshold was fed to the model.

```
Max Similarity Score for the True Positive Class: 100.0
Max Similarity Score for the False Negative Class: 49.58
Max Similarity Score for the Validation Similarity Scores: 100.0
Minimum Similarity Score for the True Positive Class: 50.18
Minimum Similarity Score for the False Negative Class: 0
Minimum Value for the Validation Similarity Scores: 0
Average Similarity Score for the True Positive Class: 97.49293995859215
Average Similarity Score for the False Negative Class: 21.894273995077924
Average Similarity Score for the Validation Data: 62.95217016491753
```

Figure 82: Various Inter Class Statistics from the Median Threshold.

4.10.5 Validation Data - Complement Spectrum

4.10.6 Median Percentile Threshold

In the Complement Section, a total of 3,467 experiments were carried out. This gave us a total words/tokens length of same number as each User Tweet from the years January 1, 2015 to August 22, 2022 were mined from the Twitter API and the similarity index between each Tweet and the TF-IDF Model. In this spectrum, the model achieved a True Negative rate of 58.667% and a False Positive rate of 41.332%.

It is fair to say that this model in this spectrum performs averagely while using the Median Threshold. The following figures below provide an overview of the model's performance in identifying True Negatives(User's Tweets that have no suicidal intentions) and False Positives(User's Tweets that have no suicidal intentions but the model incorrectly identifies them as having suicidal intentions).

SIMILARITY SCORE STATISTICS FOR THE DIFFERENT CLASSES

	Maximum	Minimum	Average
True Negative Rate	100.00	50.36	97.187984
False Positive Rate	49.92	0.00	21.646588
Validation Data	100.00	0.00	65.964788

Figure 83: A table showing various Descriptive Statistics from the True Negatives and False Positives when the Median Threshold was fed to the model.

```
Max Similarity Score for the True Negative Class: 100.0
Max Similarity Score for the False Positive Class: 49.92
Max Similarity Score for the Validation Similarity Scores: 100.0
Minimum Similarity Score for the True Negative Class: 50.36
Minimum Similarity Score for the False_Positive Class: 0
Minimum Value for the Validation Similarity Scores: 0
Average Similarity Score for the True Negative Class: 97.18798426745329
Average Similarity Score for the False Positive Class: 21.64658757850663
Average Similarity Score for the Validation Data: 65.9647880011538
```

Figure 84: Various Inter Class Statistics from the Median Threshold.

4.10.7 Lower Percentile Threshold

While using the Lower Percentile Threshold of 20, the model did not perform well. This is to be expected as in the Median Threshold, the model's performance was average. The model

achieved a True Negative rate of 41.33% and a False Positive Rate of 58.67%. The figures below provide more information about the model's performance at this threshold.

SIMILARITY SCORE STATISTICS FOR THE DIFFERENT CLASSES

	Maximum	Minimum	Average
True Negative Rate	49.92	0.00	21.646588
False Positive Rate	100.00	50.36	97.187984
Validation Data	100.00	0.00	65.964788

Figure 85: A table showing various Descriptive Statistics from the True Positive and False Negative Class.

```
Max Similarity Score for the True Negative Class: 49.92
Max Similarity Score for the False Positive Class: 100.0
Max Similarity Score for the Validation Similarity Scores: 100.0
Minimum Similarity Score for the True Negative Class: 0
Minimum Similarity Score for the False_Positive Class: 50.36
Minimum Value for the Validation Similarity Scores: 0
Average Similarity Score for the True Negative Class: 21.64658757850663
Average Similarity Score for the False Positive Class: 97.18798426745329
Average Similarity Score for the Validation Data: 65.9647880011538
```

Figure 86: Various Inter Class Statistics from the Lower Percentile Threshold.

4.10.8 Upper Percentile Threshold

In these sets of experiments, the goal was to test how effective the model is at detecting True Negatives and False Positives. The results however were very disappointing while using the Upper Percentile Threshold as the model had a True Negative rate of 0.029% and False Positive rate of 99.971%. Intuitively, it means that at this threshold level, the TF-IDF model's results become insignificant. More of the choice would be discussed extensively in Chapter 5 but this sets of experiments are important for the sake of satisfying curiosity and checking the limits of the model.

SIMILARITY SCORE STATISTICS FOR THE DIFFERENT CLASSES

	Maximum	Minimum	Average
True Negative Rate	0.0	0.0	0.000000
False Positive Rate	100.0	100.0	100.000000
Validation Data	100.0	0.0	99.971157

Figure 87: A table showing various Descriptive Statistics from the True Negatives and False Positives when the Upper Percentile Threshold was fed to the model.

```
Max Similarity Score for the True Negative Class: 0
Max Similarity Score for the False Positive Class: 100.0
Max Similarity Score for the Validation Similarity Scores: 100.0
Minimum Similarity Score for the True Negative Class: 0
Minimum Similarity Score for the False_Positive Class: 100.0
Minimum Value for the Validation Similarity Scores: 0
Average Similarity Score for the True Negative Class: 0.0
Average Similarity Score for the False Positive Class: 100.0
Average Similarity Score for the Validation Data: 99.97115661955581
```

Figure 88: Various Inter Class Statistics from the Median Threshold.

4.10.9 Validation Data - Similar Spectrum

4.10.10 Median Percentile Threshold

In the Similar Spectrum, a total of 3,770 experiments were carried out. The Median Threshold of 50 was fed to the model and the TF-IDF model achieved fair results with a True Positive Rate of 47.19% and a False Negative Rate of 52.81%. The figures below provide a Visual Summary of the experiments.

SIMILARITY SCORE STATISTICS FOR THE DIFFERENT CLASSES

	Maximum	Minimum	Average
True Positive Rate	100.00	50.07	94.325745
False Negative Rate	49.96	0.00	28.826052
Validation Data	100.00	0.00	59.734263

Figure 89: A table showing various Descriptive Statistics from the True Positives and False Negatives when the Median Threshold was fed to the model.

```
Max Similarity Score for the True Positive Class: 100.0
Max Similarity Score for the False Negative Class: 49.96
Max Similarity Score for the Validation Similarity Scores: 100.0
Minimum Similarity Score for the True Positive Class: 50.07
Minimum Similarity Score for the False_Negative Class: 0
Minimum Value for the Validation Similarity Scores: 0
Average Similarity Score for the True Positive Class: 94.32574480044966
Average Similarity Score for the False Negative Class: 28.826052235057777
Average Similarity Score for the Validation Data: 59.73426259946958
```

Figure 90: Various Inter Class Statistics from the Median Threshold.

4.10.11 Lower Percentile Threshold

As expected in the Lower Percentile Threshold, the model's performance remarkably increases. The True Positive rate jumps to 92.9% and the False Negative rate drops to 7.08%. The figures below provide further clarification of the model's performance.

SIMILARITY SCORE STATISTICS FOR THE DIFFERENT CLASSES

	Maximum	Minimum	Average
True Positive Rate	100.00	20.0	63.000825
False Negative Rate	19.99	0.0	16.877453
Validation Data	100.00	0.0	59.734263

Figure 91: A table showing various Descriptive Statistics from the True Positives and False Negatives in the Similar Spectrum.

```
Max Similarity Score for the True Positive Class: 100.0
Max Similarity Score for the False Negative Class: 19.99
Max Similarity Score for the Validation Similarity Scores: 100.0
Minimum Similarity Score for the True Positive Class: 20.0
Minimum Similarity Score for the False_Negative Class: 0
Minimum Value for the Validation Similarity Scores: 0
Average Similarity Score for the True Positive Class: 63.00082500713681
Average Similarity Score for the False Negative Class: 16.877453183520604
Average Similarity Score for the Validation Data: 59.73426259946958
```

Figure 92: Various Inter Class Statistics from the Lower Percentile Threshold.

4.10.12 Upper Percentile Threshold

As expected in the Upper Percentile Threshold, the model suffers. This is intuitive as if in the Median Threshold the model performs fairly, it would perform a bit worse in the Upper Percentile Threshold. When feeding this threshold to the model, the model had a True Positive Rate of 41.78%, which means that the model could correctly identify Suicidal Intents in User's Tweets, 42%(approximated) of the time and a high False Negative Rate of 58.22%. The figures below provide more summary of the model's performance.

SIMILARITY SCORE STATISTICS FOR THE DIFFERENT CLASSES

	Maximum	Minimum	Average
True Positive Rate	100.00	75.05	98.885771
False Negative Rate	74.81	0.00	31.641494
Validation Data	100.00	0.00	59.734263

Figure 93: A table showing various Descriptive Statistics from the True Negatives and False Positives when the Upper Percentile Threshold was fed to the model.

```
Max Similarity Score for the True Positive Class: 100.0
Max Similarity Score for the False Negative Class: 74.81
Max Similarity Score for the Validation Similarity Scores: 100.0
Minimum Similarity Score for the True Positive Class: 75.05
Minimum Similarity Score for the False_Negative Class: 0
Minimum Value for the Validation Similarity Scores: 0
Average Similarity Score for the True Positive Class: 98.88577142857146
Average Similarity Score for the False Negative Class: 31.64149430523926
Average Similarity Score for the Validation Data: 59.73426259946958
```

Figure 94: Various Inter Class Statistics from the Median Threshold.

4.11 Building the BERT Model

4.11.1 Validation Data

For the Validation Data - Complement Spectrum, the process also involved mining the Twitter API for fresh data different from the MINI data. The same methodology was used as was for the Validation Data above, the only difference this time is the search query. The purpose is to test the model using a search query that is at the opposite end of the spectrum of 'self harm' which is 'love'. The actually search term used to extract this data was 'i am in love'.

In calculating the Similarity Score in this spectrum, a different threshold of 0.37 - Maximum Value of the Similarity Scores Statistic - was chosen. Also the logic changed, because in this spectrum we are interested in the True Negative Class and the False Positive Class. That is, how well can the model spot User Tweets that don't contain Suicidal Intent? So instead of populating the True Negative Class with Similarity Scores greater than or equal to chosen threshold, we would rather populate it with Similarity Scores less than or equal to the Maximum Threshold, i.e the maximum number for similarity scores.

In the Complement Spectrum, the total number of tokens was 3467 and these tweets spanned a total of 7 years from January 1, 2015 to August 12, 2022. The figure below shows a tokenized User Tweet after pre-processing.

```
['liveindizilan',
 'and',
 'at',
 'the',
 'last',
 'i',
 'ove',
 'you',
 'fandomi',
 'love',
 'you',
 'with',
 'my',
 'whole',
 'heart',
 'and',
 'i',
 'am',
 'so',
 'happy',
 'that',
 'i',
 'am',
 'here',
 'with']
```

Figure 95: A sample tweet from the Validation Data - Complement Spectrum. The stop words haven't been removed yet.

Since this Validation Data is for testing how well the model recognizes True Negatives, we are to compute two Model Performance metrics - True Negative Rate and False Positive Rate.

The table below provides a summary of various Descriptive Statistics for the True Negative Rate, False Positive Rate and the overall Validation Similarity Scores.

SIMILARITY SCORE STATISTICS - COMPLEMENT SPECTRUM

	Maximum	Minimum	Average
True Negative Rate	0.37	0.00	0.262940
False Positive Rate	0.41	0.41	0.410000
Validation Data	0.41	0.00	0.272059

Figure 96: An image showing a summary of the Maximum, Minimum and Average Values of the True Negatives, False Positives and as well as the total Validation Data gotten from the Validation Data - Complement Spectrum.

The total amount of tokens in the True Negative Class was 3252 out of 3467 total tokens. This means based on the Validation Data - Complement Spectrum, the model has a True Negative Rate of 93.8% and a False Positive Rate of 6.20%.

4.11.2 Validation Data - Complement Spectrum

For the Validation Data - Complement Spectrum, the process also involved mining the Twitter API for fresh data different from the MINI data. The same methodology was used as was for the Validation Data above, the only difference this time is the search query. The purpose is to test the model using a search query that is at the opposite end of the spectrum of 'self harm' which is 'love'. The actually search term used to extract this data was 'i am in love'.

In calculating the Similarity Score in this spectrum, a different threshold of 0.37 - Maximum Value of the Similarity Scores Statistic - was chosen. Also the logic changed, because in this spectrum we are interested in the True Negative Class and the False Positive Class. That is, how well can the model spot User Tweets that don't contain Suicidal Intent? So instead of populating the True Negative Class with Similarity Scores greater than or equal to chosen threshold, we would rather populate it with Similarity Scores less than or equal to the Maximum Threshold, i.e the maximum number for similarity scores.

In the Complement Spectrum, the total number of tokens was 3467 and these tweets spanned a total of 7 years from January 1, 2015 to August 12, 2022. The figure below shows a tokenized User Tweet after pre-processing.

```
['liveindizilan',
 'and',
 'at',
 'the',
 'last',
 'i',
 'ove',
 'you',
 'fandomi',
 'love',
 'you',
 'with',
 'my',
 'whole',
 'heart',
 'and',
 'i',
 'am',
 'so',
 'happy',
 'that',
 'i',
 'am',
 'here',
 'with']
```

Figure 97: A sample tweet from the Validation Data - Complement Spectrum. The stop words haven't been removed yet.

Since this Validation Data is for testing how well the model recognizes True Negatives, we are to compute two Model Performance metrics - True Negative Rate and False Positive Rate. The table below provides a summary of various Descriptive Statistics for the True Negative Rate, False Positive Rate and the overall Validation Similarity Scores.

SIMILARITY SCORE STATISTICS - COMPLEMENT SPECTRUM

	Maximum	Minimum	Average
True Negative Rate	0.37	0.00	0.262940
False Positive Rate	0.41	0.41	0.410000
Validation Data	0.41	0.00	0.272059

Figure 98: An image showing a summary of the Maximum, Minimum and Average Values of the True Negatives, False Positives and as well as the total Validation Data gotten from the Validation Data - Complement Spectrum.

The total amount of tokens in the True Negative Class was 3252 out of 3467 total tokens. This means based on the Validation Data - Complement Spectrum, the model has a True Negative Rate of 93.8% and a False Positive Rate of 6.20%.

4.11.3 Validation Data - Similar Spectrum

For the Validation Data - Complement Spectrum, the process also involved mining the Twitter API for fresh data different from the MINI data. The same methodology was used as was for the Validation Data above, the only difference this time is the search query. The purpose is to test the model using a search query that is at the opposite end of the spectrum of 'self harm' which is 'love'. The actually search term used to extract this data was 'i am in love'.

In calculating the Similarity Score in this spectrum, a different threshold of 0.37 - Maximum Value of the Similarity Scores Statistic - was chosen. Also the logic changed, because in this spectrum we are interested in the True Negative Class and the False Positive Class. That is, how well can the model spot User Tweets that don't contain Suicidal Intent? So instead of populating the True Negative Class with Similarity Scores greater than or equal to chosen threshold, we would rather populate it with Similarity Scores less than or equal to the Maximum Threshold, i.e the maximum number for similarity scores.

In the Complement Spectrum, the total number of tokens was 3467 and these tweets spanned a total of 7 years from January 1, 2015 to August 12, 2022. The figure below shows a tokenized User Tweet after pre-processing.

```
['liveindizilan',
 'and',
 'at',
 'the',
 'last',
 'i',
 'ove',
 'you',
 'fandomi',
 'love',
 'you',
 'with',
 'my',
 'whole',
 'heart',
 'and',
 'i',
 'am',
 'so',
 'happy',
 'that',
 'i',
 'am',
 'here',
 'with']
```

Figure 99: A sample tweet from the Validation Data - Complement Spectrum. The stop words haven't been removed yet.

Since this Validation Data is for testing how well the model recognizes True Negatives, we are to compute two Model Performance metrics - True Negative Rate and False Positive Rate. The table below provides a summary of various Descriptive Statistics for the True Negative Rate, False Positive Rate and the overall Validation Similarity Scores.

SIMILARITY SCORE STATISTICS - COMPLEMENT SPECTRUM

	Maximum	Minimum	Average
True Negative Rate	0.37	0.00	0.262940
False Positive Rate	0.41	0.41	0.410000
Validation Data	0.41	0.00	0.272059

Figure 100: An image showing a summary of the Maximum, Minimum and Average Values of the True Negatives, False Positives and as well as the total Validation Data gotten from the Validation Data - Complement Spectrum.

The total amount of tokens in the True Negative Class was 3252 out of 3467 total tokens. This means based on the Validation Data - Complement Spectrum, the model has a True Negative Rate of 93.8% and a False Positive Rate of 6.20%.

4.12 Ensemble Model

4.12.1 Validation Data

4.12.2 Median Threshold

In the Ensemble Model, the output for the Jaccard and TF-IDF models were combined and then their average was used as the Similarity Index. The True Positive rate in the Median Threshold was 51.349% and the False Negative Rate was 48.651%. The figures below show the other statistics of the model based on the Validation Data.

SIMILARITY SCORE STATISTICS FOR THE DIFFERENT CLASSES

	Maximum	Minimum	Average
True Positive Rate	50.000	40.075	49.667734
False Negative Rate	39.965	0.000	12.275347
Validation Data	50.000	0.000	31.476085

Figure 101: A table showing various Descriptive Statistics from the True Positives and False Negatives.

```
Max Similarity Score for the True Positive Class: 50.0
Max Similarity Score for the False Negative Class: 39.965
Max Similarity Score for the Validation Similarity Scores: 50.0
Minimum Similarity Score for the True Positive Class: 40.075
Minimum Similarity Score for the False Negative Class: 0.0
Minimum Value for the Validation Similarity Scores: 0.0
Average Similarity Score for the True Positive Class: 49.667733576642334
Average Similarity Score for the False Negative Class: 12.275346687211082
Average Similarity Score for the Validation Data: 31.476085082458766
```

Figure 102: Various Inter Class Statistics from the Median Threshold.

4.12.3 Lower Percentile Threshold

While feeding the Lower Percentile Threshold to the model, it achieved a True Positive rate of 56.11% and a False Negative Rate of 43.89%. This means that 25% of the time, the Ensemble Model at the Lower Percentile Range, would be able to identify Suicidal Intentions correctly

from User's Tweets. But this also means that 25% of the time, the model would incorrectly label User's Tweets that contain Suicidal Intentions as negative/not belonging to the class rather than positive. The figures below, as has been the case in previous sections provides a Visual Summary of the experiments.

SIMILARITY SCORE STATISTICS FOR THE DIFFERENT CLASSES

	Maximum	Minimum	Average
True Positive Rate	50.00	20.025	47.885257
False Negative Rate	19.85	0.000	10.498689
Validation Data	50.00	0.000	31.476085

Figure 103: A table showing various Descriptive Statistics from the True Positives and False Negatives.

```
Max Similarity Score for the True Positive Class: 50.0
Max Similarity Score for the False Negative Class: 19.85
Max Similarity Score for the Validation Similarity Scores: 50.0
Minimum Similarity Score for the True Positive Class: 20.025
Minimum Similarity Score for the False Negative Class: 0.0
Minimum Value for the Validation Similarity Scores: 0.0
Average Similarity Score for the True Positive Class: 47.88525718102874
Average Similarity Score for the False Negative Class: 10.498689154568748
Average Similarity Score for the Validation Data: 31.476085082458766
```

Figure 104: Various Inter Class Statistics from the Lower Percentile Threshold.

4.12.4 Upper Percentile Threshold

In the Upper Percentile Threshold, the Ensemble Model could not predict one outcome correctly i.e the amount of True Positives was 0 implying that the False Negative was 100%.

4.12.5 Validation Data - Complement Spectrum

4.12.6 Median Threshold

Combining the power of the Jaccard and TF-IDF Models yielded impressive results in identifying True Negatives. In the Median Threshold, the model achieved a perfect score of 100% in this spectrum, correctly identifying all the True Negatives(3467 out of 3467) in this data spectrum. The figures below provide more Visual cues as to the nature of the experiments.

SIMILARITY SCORE STATISTICS - COMPLEMENT SPECTRUM

	Maximum	Minimum	Average
True Negative Rate	50.205	0.0	35.38071
False Positive Rate	0.000	0.0	0.00000
Validation Data	50.205	0.0	35.38071

Figure 105: A table showing various Descriptive Statistics from the True Positives and False Negatives.

```
Max Similarity Score for the True Negative Class: 50.205
Max Similarity Score for the False Positive Class: 0
Max Similarity Score for the Validation Similarity Scores: 50.205
Minimum Similarity Score for the True Negative Class: 0.0
Minimum Similarity Score for the False Positive Class: 0
Minimum Value for the Validation Similarity Scores: 0.0
Average Similarity Score for the True Negative Class: 35.380709547158816
Average Similarity Score for the False Positive Class: 0
Average Similarity Score for the Validation Data: 35.380709547158816
```

Figure 106: Various Inter Class Statistics from the Median Threshold.

4.12.7 Lower Percentile Threshold

While passing the Lower Percentile Threshold to the Ensemble Model in the Complement Spectrum, the model achieved a True Negative Rate of 34.381% and a False Positive Rate of 65.619%. This means that in this threshold, the model has a low Specificity Rate. The figures below provide further information about the results of the experiments.

SIMILARITY SCORE STATISTICS - COMPLEMENT SPECTRUM

	Maximum	Minimum	Average
True Negative Rate	19.990	0.00	14.735747
False Positive Rate	50.205	20.03	46.197763
Validation Data	50.205	0.00	35.380710

Figure 107: A table showing various Descriptive Statistics from the True Negatives and False Positives.

```
Max Similarity Score for the True Negative Class: 19.99
Max Similarity Score for the False Positive Class: 50.205
Max Similarity Score for the Validation Similarity Scores: 50.205
Minimum Similarity Score for the True Negative Class: 0.0
Minimum Similarity Score for the False Positive Class: 20.03
Minimum Value for the Validation Similarity Scores: 0.0
Average Similarity Score for the True Negative Class: 14.735746644295297
Average Similarity Score for the False Positive Class: 46.19776263736252
Average Similarity Score for the Validation Data: 35.380709547158816
```

Figure 108: Various Inter Class Statistics from the Lower Percentile Threshold.

4.12.8 Upper Percentile Threshold

The same results were achieved when the Upper Percentile Threshold was passed to the Ensemble Model. The model achieved a perfect True Negative Rate and a perfect False Positive rate. This means that the Ensemble Model in the Complement Spectrum with a threshold value of 75, has a perfect Specificity Rate.

Figures below provide more Visual Cues similar to the those in the Lower Percentile Threshold:

SIMILARITY SCORE STATISTICS - COMPLEMENT SPECTRUM

	Maximum	Minimum	Average
True Negative Rate	50.115	0.0	50.062204
False Positive Rate	0.000	0.0	0.000000
Validation Data	50.115	0.0	50.062204

Figure 109: A table showing various Descriptive Statistics from the True Positives and False Negatives.

```
Max Similarity Score for the True Negative Class: 50.115
Max Similarity Score for the False Positive Class: 0
Max Similarity Score for the Validation Similarity Scores: 50.115
Minimum Similarity Score for the True Negative Class: 0.0
Minimum Similarity Score for the False Positive Class: 0
Minimum Value for the Validation Similarity Scores: 0.0
Average Similarity Score for the True Negative Class: 50.06220363426467
Average Similarity Score for the False Positive Class: 0
Average Similarity Score for the Validation Data: 50.06220363426467
```

Figure 110: Various Inter Class Statistics from the Upper Percentile Threshold.

4.12.9 Validation Data - Similar Spectrum

4.12.10 Median Threshold

The Ensemble Model achieved a True Positive Rate of 41.194% and a False Negative Rate of 58.806%. This means that the model has a High Sensitivity Rate. It equally means that the Ensemble Model with a Median Threshold does not perform well in identifying User's Tweets with Suicidal Intent. The figures below provide more information about the experiments carried out here:

SIMILARITY SCORE STATISTICS FOR THE DIFFERENT CLASSES

	Maximum	Minimum	Average
True Positive Rate	50.00	40.095	49.595940
False Negative Rate	39.85	0.000	16.047176
Validation Data	50.00	0.000	29.867131

Figure 111: A table showing various Descriptive Statistics from the True Positives and False Negatives.

```
Max Similarity Score for the True Positive Class: 50.0
Max Similarity Score for the False Negative Class: 39.85
Max Similarity Score for the Validation Similarity Scores: 50.0
Minimum Similarity Score for the True Positive Class: 40.095
Minimum Similarity Score for the False Negative Class: 0.0
Minimum Value for the Validation Similarity Scores: 0.0
Average Similarity Score for the True Positive Class: 49.595940115904725
Average Similarity Score for the False Negative Class: 16.0471763644565
Average Similarity Score for the Validation Data: 29.86713129973479
```

Figure 112: Various Inter Class Statistics from the Median Threshold.

4.12.11 Lower Percentile Threshold

By reducing the threshold from the Median to the Lower Percentile, as expected the model's performance improved albeit incrementally. In the Lower Percentile Threshold, the model achieved a True Positive Rate of 52.653% and a False Negative Rate of 47.347%. The figures below tells us more about the experiments performed here:

SIMILARITY SCORE STATISTICS FOR THE DIFFERENT CLASSES

	Maximum	Minimum	Average
True Positive Rate	50.000	20.02	44.598290
False Negative Rate	19.995	0.00	13.485423
Validation Data	50.000	0.00	29.867131

Figure 113: A table showing various Descriptive Statistics from the True Negatives and False Positives.

```
Max Similarity Score for the True Positive Class: 50.0
Max Similarity Score for the False Negative Class: 19.995
Max Similarity Score for the Validation Similarity Scores: 50.0
Minimum Similarity Score for the True Positive Class: 20.02
Minimum Similarity Score for the False Negative Class: 0.0
Minimum Value for the Validation Similarity Scores: 0.0
Average Similarity Score for the True Positive Class: 44.59828967254408
Average Similarity Score for the False Negative Class: 13.485422969187656
Average Similarity Score for the Validation Data: 29.86713129973479
```

Figure 114: Various Inter Class Statistics from the Lower Percentile Threshold.

4.12.12 Upper Percentile Threshold

While feeding the model with the Upper Percentile Threshold, the model achieved a 100% False Negative Rate, which means it couldn't predict any of the positive samples as actually positive. A this threshold, the Ensemble Model loses any value left and thus is not relevant to this spectrum dataset and ergo to real world data and applications.

4.13 Interpretation of Results

4.13.1 Validation Results

In the Validation Spectrum, these involved a set of experiments whereby we took various sample User Tweets from the January 1, 2015 to 22, August 2022, then tested the similarity between each of these tweets to the SWEETS MINI using the Median Threshold as the cut-off mark. The table below shows a summary of the performance of the different Document Similarity Algorithms at this spectrum.

VALIDATION SPECTRUM WITH THE MEDIAN THRESHOLD

	Jaccard	TF-IDF	ENSEMBLE	BERT*
True Positive Rate	84.2200	54.3100	51.349	91.667
False Negative Rate	15.7800	45.6900	48.651	8.333
Recall	0.8422	0.5431	1.000	1.0
Accuracy Score	84.2200	54.3100	51.349	91.667
Experiments Count	2668.0000	2668.0000	2668.000	12
Computation Time (ms)	23603.9150	616172.3850	640956.478	0*
Threshold	0.2800	50.0000	40.000	30

Figure 115: A table showing the performance of the different Document Similarity Algorithms in the Validation Spectrum.

As can be seen from the table, the BERT Model performed best in 4 categories - True Positive Rate(TPR), False Negative Rate(FNR), Recall and Accuracy Score. The Recall statistic is especially noteworthy as the BERT Model achieved a perfect score in this Classification Performance Metric. However, we must note that the sample size or Experiments count was low for BERT compared to other models' sample size like Jaccard for instance which was 2,668. The reason for this was because though the BERT Model may be powerful in his predictive ability, a major shortcoming is it's Computation Time. The figure below shows the how long it took for each model to perform their Classification task:

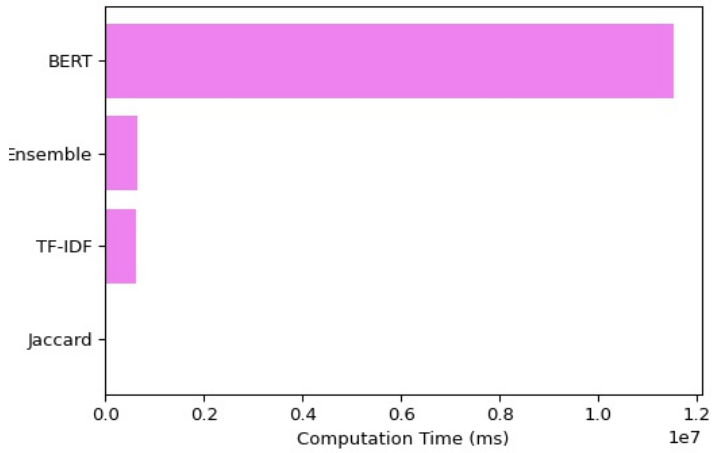

Figure 116: A Horizontal Bar Chart showing the Computation Time of the different Models

We can see from the figure above that despite having the smallest sample size/experiments count, the BERT Model still supersedes all the other models combined in Computation Time. This means that despite it's strong classification performance, albeit with a small sample size, it may be arcane to use this model in production. The figure below charts the various models against their experiments count for us to have a panoramic view of how computationally expensive the BERT Model would be in production:

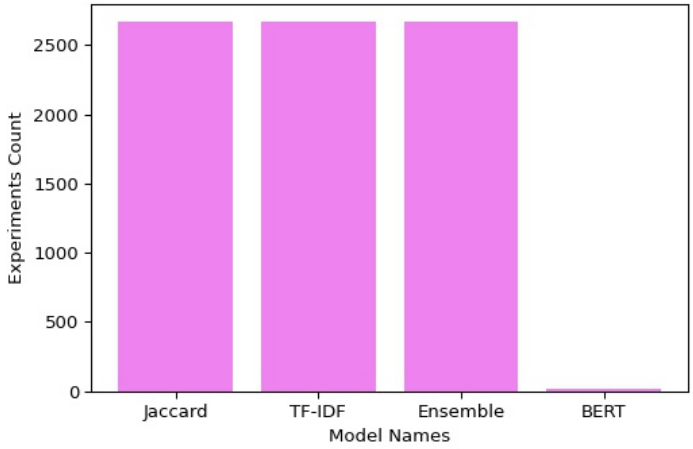

Figure 117: A Bar Chart showing the different models and their experiments count. In the Validation Spectrum, the Data used was gotten from 7years and a total of 8,016 experiments were carried out in this spectrum in order to validate and test the SWEETS MINI.

Since, the BERT Model is computationally expensive, the next best model was the Jaccards'. The model came second in in all other Classification metrics except the Recall where it was beaten by the Ensemble Model(a combination of the Jaccard and the TF-IDF). These metrics include a healthy True Positive Rate, which means that this model can correctly identify Suicidal Tendencies in User's Tweets 84.22% of the time e.g if there are 1000 User Tweets with Suicidal Tendencies, the Jaccard Model would correctly identify 842 of them. Inversely, with a False Negative Rate of 15.780%, the model would not identify 158 of those User's Tweets with Suicidal Tendencies. While this is a good springboard, improvements can still be made. This would be discussed in Chapter 5.

It is also important to note that the Threshold point used for the Jaccard Model in the Validation Spectrum was 0.28, this was arrived at as the most optimum threshold after a series of experiments with two other threshold values - 0.15 and 0.37. This threshold value provided the highest Recall Rate and hence would be adopted as the Model to be used to classify if a User's Tweets contains Suicidal Tendencies or not.

4.13.2 Similar Spectrum Results

In the Similar Spectrum, the aim was to select data that was similar to the Validation Data but different in content. In essence, similar themes but different content and sometimes context. This data was mined from the Twitter API in a span of 7 years from January 1, 2015 to August 22, 2022. In this spectrum, the Classification Metrics that concern us are similar to those of the Validation Spectrum - True Positive Rate, False Negative Rate, Accuracy Score. The figure below shows the different models and their Classification performance using the different rates as a metric guide.

VALIDATION - SIMILAR SPECTRUM WITH THE MEDIAN THRESHOLD

	Jaccard	TF-IDF	ENSEMBLE	BERT*
True Positive Rate	99.73500	47.188	41.194	100.0
False Negative Rate	0.26500	52.812	58.806	0.0
Recall	0.99735	0.472	1.000	1.0
Accuracy Score	99.73500	47.188	41.194	100.0
Experiments Count	3770.00000	3770.000	3770.000	25.0
Computation Time (ms)	28002.35900	1010864.712	827789.851	21600000.0
Threshold	0.37000	50.000	40.000	30.0

Figure 118: A table showing the performance of the different Document Similarity Algorithms in the Validation Spectrum.

As we can see from the figure above, the BERT model once again outperforms the other models but it's Computation Time dwarfs the other models and it's experiments count is too small. The next best model is the Jaccard which achieves a high Accuracy Score of 99.735%; which means the Jaccard Model is able to recognise similar documents or practically similar User Tweets with suicidal tendencies - this can be for those instances whereby the situation has not become severe but there are elements of suicidal intent in the tweets.

The figure below shows various classification metrics of the different models:

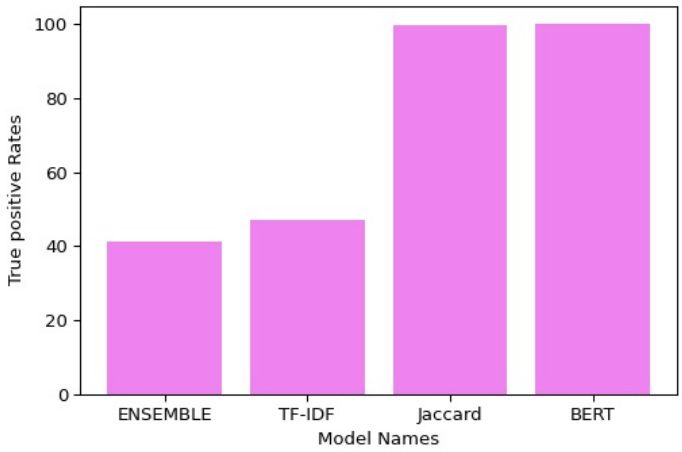

Figure 119: A Bar Chart showing the True Positive Rates(TPR) of the different Models. One major visual cue is the Jaccard and BERT Model having nearly similar rates despite the latter's smaller sample size.

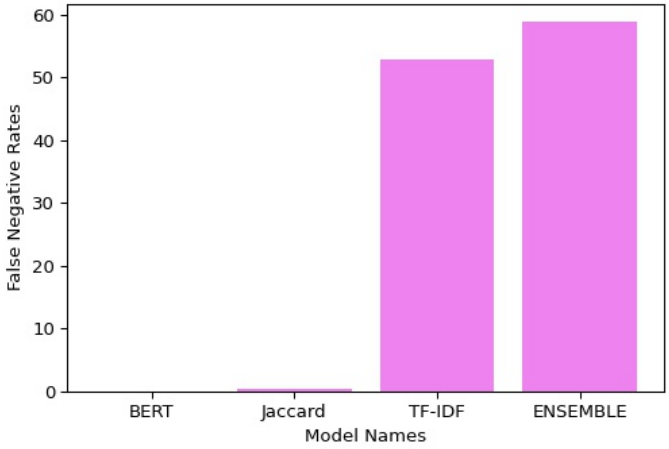

Figure 120: A Bar Chart showing the False Negative Rates of the different Models. In this metric, the smaller the FNR is, the better and more valid the model's results and predictions would be.

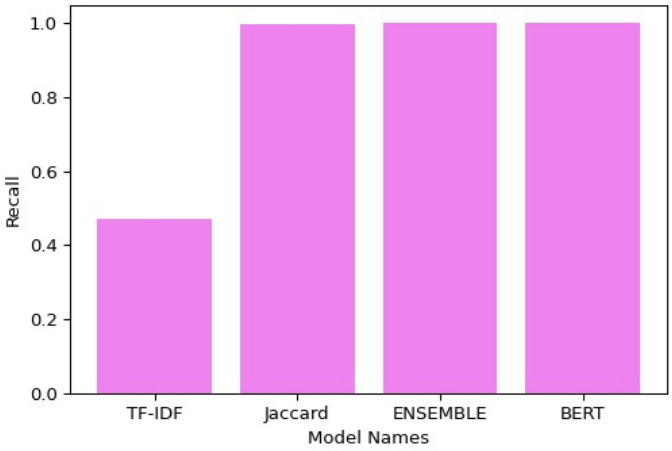

Figure 121: A Bar Chart showing the Recall Rates of the different Models. The Jaccard, BERT and Ensemble achieve high scores in this metric.

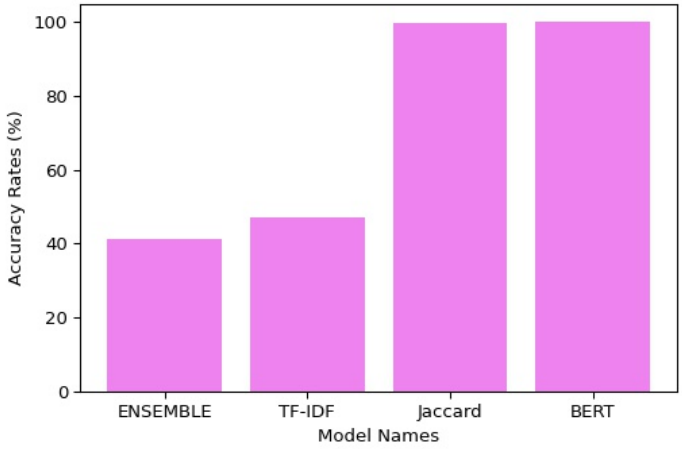

Figure 122: A Bar Chart showing the Accuracy Score of the different Models. The Jaccard and BERT and the best models when it comes to accuracy. But because of BERT'S small experiments size, the Jaccard Model also wins in this spectrum.

4.13.3 Complement Spectrum Results

The Complement Spectrum involved comparing the SWEETS MINI to data that is a complement to it. In essence, we want to test and experiment how effective each of the ,models are in identifying dissimilar documents. In this case, we want to be able to test the efficacy and efficiency of the model in determining User's Tweets without Suicidal Tendencies. The figure below provides an overview of how the models performed in this spectrum.

VALIDATION - COMPLEMENT SPECTRUM WITH THE MEDIAN THRESHOLD

	Jaccard	TF-IDF	ENSEMBLE	BERT*
True Negative Rate	93.799	58.667	100.000	100.0
False Positive Rate	6.201	41.333	0.000	0.0
Accuracy Score	93.799	58.667	100.000	100.0
Experiments Count	3467.000	3467.000	3467.000	10.0
Computation Time (ms)	25955.301	1010864.712	1690820.646	15480000.0
Threshold	0.370	50.000	55.000	30.0

Figure 123: A table showing the performance of the different Document Similarity Algorithms in the Complement Spectrum.

In the figure above, we can see that in the True Negative Rate, the model's ability to accurately classify User's Tweets that don't contain Suicidal Tendencies, there were all able to do this well. The BERT and ENSEMBLE models achieved a perfect score and the Jaccard Model achieved a True Negative Rate(TNR) of 93.799%. Intuitively, the BERT and ENSEMBLE models achieved a False Positive Rate(FPR) of 0% and the Jaccard Model achieved a 6.201%. Going back to our earlier example, if there are a 1000 User's Tweets that don't contain Suicidal Tendencies, the BERT and ENSEMBLE Model would accurately classify all of them - 1000 - as not having Suicidal Tendencies while the Jaccard Model would classify them correctly 93.799% of the time i.e 937 out of 1000; and lastly the TF-IDF Model would be able to classify them 58.667% of the time i.e 586 out of 1000.

In the figures below, we chart the various Classification Metrics to have a more nuanced Visual Cue and feel of the Classification Performance of each of the models in this spectrum.

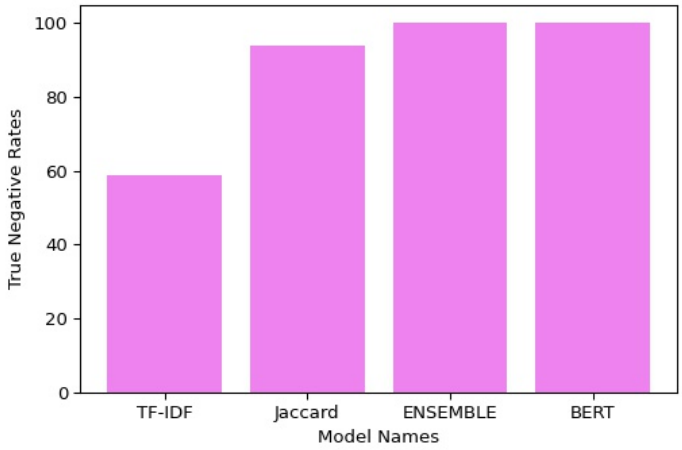

Figure 124: A Bar Chart showing the True Negative Rates(TNR) of the different Models. Both the BERT and the ENSEMBLE Models achieve high Precision Rates in this spectrum. Making them compatible for identifying non-suicidal tendencies tweets. One use case in production could be passing the User's Tweet through this model first then if there's little to no similarity do a check that the User Tweet does not contain Suicidal Tendencies. The only issue seems to be the computation time, however, since it would be one sentence, this may not be a problem.

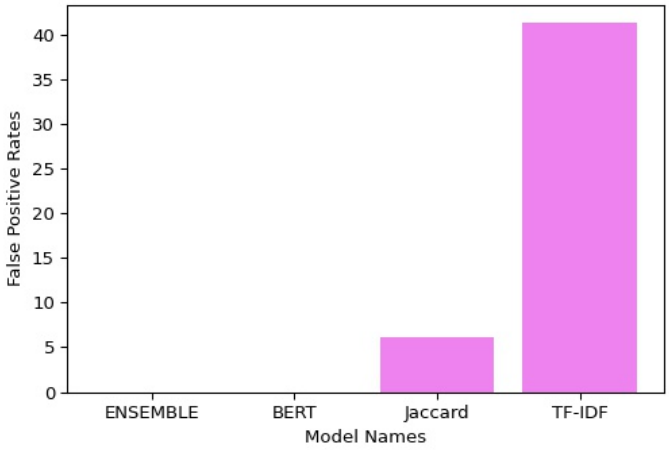

Figure 125: A Bar Chart showing the False Positive Rates of the different Models. The ENSEMBLE and BERT Model both achieved 100% in the True Negative Rates metric, therefore, there would have 0% False Positive Rates since the models were able to correctly identify non-suicidal tendencies in User's Tweets.

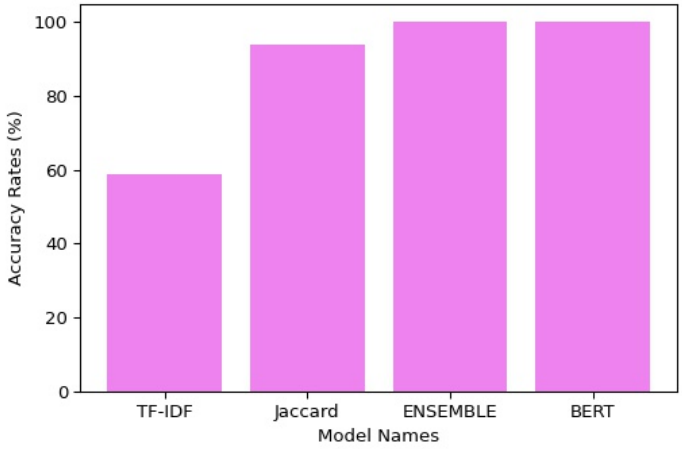

Figure 126: A Bar Chart showing the Accuracy Score of the different Models. All the models did well in this spectrum even the TF-IDF model performed reasonably well with an Accuracy Score of 58.667%. Hence the decision of which model to use would be a matter of discretion and a case by case basis. Any of the Jaccard, ENSEMBLE and BERT are very good at identifying True Negatives.

5 Summary and Conclusion

It is estimated that more than 700,000 people die due to suicide every year(who.int 2021). Its impact not only on the economy but in the primary unit of the family cannot be understated. This research seeks to help PCP better in identifying suicidal tendencies in text – especially in User Tweets since most people have an active Digital Phenotype and one can be able to be proactive in preventing suicide. But there are also concerns, one of which is do individuals have a right to end their life? This is especially relevant in the current climate where the US judiciary government is seeking to overturn an earlier decision that makes abortion legal in the USA(Josh Gerstein 2022); such a scenario can also play here ethically and morally. If I want to end my life, is it my choice? If yes, what would be the impact of this? The purpose of this research however is not euthanasia but rather to provide a model or more appropriately a tool for identifying suicidal tendencies.

In the process of building a model that can accurately predict if a User's Tweets has Suicidal Tendencies or not, four different models were used - Jaccard, TF-IDF, ENSEMBLE(Jaccard and TF-IDF) and BERT. Of this four, the Jaccard was the most durable as it was able to accurately predict True positives(84.22% in the Validation Spectrum) and True Negatives(93.799% in the Complement Spectrum) in all the spectrum of data - Validation, Similar and Complement Spectrum.The Jaccard Model also proved to be computationally inexpensive, arriving at a classification in the lowest time. However, we must also make note of the BERT Model whose performance, despite a small experimental size was perfect throughout.

A suite of Micro-Services catering to Suicidal Tendencies was hence created, from a CORPORA - SWEETS CORPORA to a CORPUS - SWEETS CORPUS to a mini version suitable for real-world deployment and applications - SWEETS MINI. With this suite of micro-services, along with the Suicidal Sentiment Lexicon(SSL), the aims and objectives of the research were hence achieved with more concomitant results and output than earlier envisaged.

5.1 Deployment

As was earlier mentioned in the Summary section, a suite of micro-products was built in order to provide researchers with robust tools to help them in undertaking further research on Suicidal Tendencies in User's Tweets. This micro-products are stand-alone products but come together to form a suite. from the Twitter Cleaning Web Application to the SWEETS CORPORA, SWEETS, CORPUS AND SWEETS MINI to the SSL.

These products would be deployed on the web as containers using Docker. Then hosted on Heroku to the web. The web applications were built with Python, Flask, HTML5, CSS3 and Google Colab.

The Github Page providing the links to the various micro-products as well as further documentation can be found here: https://github.com/laresamdeola/SWEETS

5.2 Continuous Integration and Continuous Deployment

For the Continuous Integration and Continuous Deployment(CI/CD), for this project we would be using the Github as the version control platform along with Git. Since this would be a

containerized application, Kubernetes also would be used for container orchestration - here would deal with such problems like down-times, how containers interact with each other, volumes etc.

5.3 Further Recommendation

While an Accuracy Rate of 84% achieved by the Jaccard Model in the Validation Spectrum is remarkable, it can be better. This is because, in other innocuous domains, this accuracy rate is welcome, but in a domain such as mental health and suicide, whereby the stakes are a bit higher as it can be a situation of life and death, a higher accuracy level may be required.

It is in this line of thought that the other model like BERT becomes a bit of a welcome concern, as the potential for the model to be highly accurate was shown in the small experiments but since that sample size was so small we might consider it infinitesimal; more experiments would have to be taken with the BERT model to test if it's accuracy can be as good over a larger range of validation and test data. There would be a trade-off though, either in computational resources or finance.

Also, there's the case of lexical richness; it was a bit disappointing to achieve such low lexical richness from the data mined from the Twitter API. What UI can foresee in this is that our data may fall into the trap of the echo chamber. There's some good in this and bad as well. The good is, this project and all the models, algorithms and code were all inspired from the Twitter API - from the Algorithm to Clean Twitter Data to the Suicide Sentiment Lexicon to the SWEETS CORPORA, SWEETS CORPUS and SWEETS MINI, everything was built from the foundation of data gotten from the Twitter API. this means that the models would be especially good at detecting Suicidal Tendencies in Twitter; what about outside Twitter? Should this be a concern? How adaptable can the model be? These are all valid questions and could point to areas of further research for this project.

In the end, a research project has to have a beginning and an end, it is almost impossible to cover all angles, as new problems arise everyday. Infact, it was because of this that I created a sub-section in Chapter 5 called Continuous Integration and Continuous Deployment, whereby this project is not a static being but rather an ever evolving organism that has the tendency to morph to any particular problem focus. Hence, this is a good place to start, to be able to detect Suicidal Tendencies in just Twitter and building tools to help other researchers be able to build more suicide prevention NLP products faster. The logical next step would be diving deeper into Electronic Health Records and building a suite of micro-products around EHR's regarding mental health with a focus on Suicidal Tendencies.

A Appendix

A.0.1 Methodology for the Query Words, Twitter Clean Algorithm and SWEETS CORPORA

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import nltk

nltk.download('punkt')

base_data = pd.read_csv('medinform.csv', encoding='cp1252')

"""**ANALYSE BASE DATA**"""

base_data.info()

base_data.head()

base_data.tail()

base_data['word']

frame = {
    'word': base_data['word'],
    'case_frequency': base_data['case_freq']
}

df = pd.DataFrame(frame)

print(df)

"""**DATA ENGINEERING FOR VISUAL DISPLAY OF THE DATA**"""

case_frequency = df['case_frequency']
case_frequency.describe()

highest_frequencies = [word for word in df['case_frequency'] if word > 1567]

print(len(highest_frequencies))
```

```
print(highest_frequencies)

highest_words = []

for index, each in enumerate(df['word']):
  for index2, word in enumerate(df['case_frequency']):
    if word > 1567 and index == index2:
      highest_words.append(each)
      break

print(highest_words)

print(len(highest_words))

top_50 = {
    'top words': highest_words,
    'top frequencies': highest_frequencies
}

top_50_frame = pd.DataFrame(top_50)

top_50_frame

top_50_frame = top_50_frame.sort_values('top frequencies')

top_50_frame

plt.plot(top_50_frame['top words'][:20], top_50_frame['top frequencies'][:20],
color='violet')
plt.xlabel('Top Words')
plt.ylabel('Top Words Frequencies')
plt.xticks(top_50_frame['top words'][:20], rotation=65)
#plt.margins(0.001)
#plt.savefig('Percentile_lineplot_better_visuals.jpg', dpi=95, orientation='landscape')
plt.show()

plt.bar(top_50_frame['top words'][:20], top_50_frame['top frequencies'][:20],
color='violet')
plt.xlabel('Top Words')
plt.ylabel('Top Words Frequencies')
plt.xticks(top_50_frame['top words'][:20], rotation=65)
#plt.margins(0.001)
```

```
#plt.savefig('Percentile_vbars_better_visuals.jpg', dpi=95, orientation='portrait')
plt.show()

plt.barh(top_50_frame['top words'][:15], top_50_frame['top frequencies'][:15],
color='violet', height=0.5)
plt.xlabel('Top Words')
plt.ylabel('Top Words Frequencies')
#plt.xticks(top_50_frame['top words'][:20], rotation=65)
#plt.margins(0.001)
#plt.savefig('Percentile_hbars_better_visuals.jpg', dpi=95, orientation='landscape')
plt.show()

from wordcloud import WordCloud, STOPWORDS

#convert dataframe to a string to input into the WordCloud Class

words_strings = ' '.join(highest_words)
stopwords = set(STOPWORDS)

print(words_strings)

#print(type(corpus_data))

wordcloud = WordCloud(width = 1400, height = 800, stopwords = stopwords,
min_font_size = 10).generate(corpus_data)

plt.figure(figsize = (12, 8), facecolor = None)
plt.imshow(wordcloud)
#plt.savefig('Corpus Word Cloud.jpg', dpi=95, orientation='landscape')
plt.show()

"""**PREPROCESSING TWITTER DATA SAMPLE**"""

from nltk.tokenize import TweetTokenizer, word_tokenize, sent_tokenize
import re
import os

data = ''

def clean_twitter_text(text):
  with open(text,"r") as read_object:
    lines = read_object.read()
```

```
    #remove special characters and punctuations
    #lines = re.sub('[\n]', '', lines)
    lines = re.sub('#', '', lines)
    lines = re.sub('\\n\\n', '', lines)
    lines = re.sub('[\\n\\n]', '', lines)
    lines = re.sub('(\n\n)', '', lines)
    lines = re.sub('[{}:_@\[\]0-9,%&*""?!/-]', '', lines)
    #remove the id and text tag
    lines = re.sub('(id)', '', lines)
    lines = re.sub('(text)', '', lines)
    lines = re.sub('(RT)', '', lines)
    #remove paragraph space/indentation
    lines = re.sub('  ', '', lines)
    #lines = re.sub(str(escape), '', lines)
    return lines

data = clean_twitter_text('Twitter Suicide Corpora.txt')

corpus_data = clean_twitter_text("Twitter Suicide Corpus.txt")

mini = clean_twitter_text('Twitter Suicide Corpora Mini.txt')

mini

mini_tokens = word_tokenize(mini)

corpora_tokens = word_tokenize(data)

new_corpora = [word for word in corpora_tokens if word.isalpha()]

corpora_strings = ' '.join(new_corpora)

corpora_tokens = word_tokenize(corpora_strings)

len(corpora_tokens)

len(set(corpora_tokens))

print(len(set(corpora_tokens)) / len(corpora_tokens))

corpora_text = nltk.Text(corpora_tokens)
```

```
len(corpora_text)

import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')

stop_words = set(stopwords.words('english'))

filter_corpora = []

for word in corpora_tokens:
  word = word.lower()
  if word not in stop_words:
    filter_corpora.append(word)

len(filter_corpora)

len(set(filter_corpora))

print(len(set(filter_corpora)) / len(filter_corpora))

corpora_frequency = nltk.FreqDist(filter_corpora)

corpora_text = nltk.Text(filter_corpora)

corpora_text.concordance('self')

corpora_text.concordance('harm')

corpora_text.concordance('goth')

corpora_text.collocation_list()

corpora_frequency.plot(25, cumulative = True)

corpora_text.dispersion_plot(['self', 'harm', 'goth', 'mental', 'life'])

corpus_tokens = word_tokenize(corpus_data)

new_corpus = [word for word in corpus_tokens if word.isalpha()]

corpus_strings = ' '.join(new_corpus)
```

```
corpus_strings

corpus_tokens = word_tokenize(corpus_strings)

len(corpus_tokens)

len(set(corpus_tokens))

print(len(set(corpus_tokens)) / len(corpus_tokens))

corpus_text = Text(corpus_tokens)

len(corpus_text)

corpus_text.concordance('self')

corpus_text.collocation_list()

corpus_text.dispersion_plot(['self', 'harm', 'mental', 'health', 'urgently',
'receive', 'repeatedly'])

corpus_text.dispersion_plot(['Needy', 'Streamer', 'goth', 'babydoll', 'health',
'care', 'counselling'])

corpus_text.dispersion_plot(['crisis', 'Gay', 'Republicans', 'especially',
'life', 'death', 'tired'])

corpus_frequency = nltk.FreqDist(corpus_text)

corpus_frequency.most_common(20)

filter_corpus = []

for word in corpus_tokens:
  word = word.lower()
  if word not in stop_words:
    filter_corpus.append(word)

len(filter_corpus)

len(set(filter_corpus))
```

```
print((len(set(filter_corpus)) / len(filter_corpus) * 100))

corpus_frequency_2 = nltk.FreqDist(filter_corpus)

corpus_frequency_2.plot(25, cumulative=True)

#mini_tokens

new_mini = [word for word in mini_tokens if word.isalpha()]

mini_strings = ' '.join(new_mini)

mini_strings

mini_tokens = word_tokenize(mini_strings)

print(len(mini_tokens))

#mini_tokens

"""ANALYSING THE MINI"""

from nltk.text import Text
import nltk.corpus

#i have to remove the stop words

text = Text(mini_tokens)

print(type(text))

text

text.concordance('goth')

text.similar('harm')

text.similar('self')

text.similar('kill')
```

```
text.similar('live')

text.similar('life')

text.common_contexts(["Streamer", "Overload"])

text.dispersion_plot(["self", "harm", "mental", "health", "risk"])

text.dispersion_plot(["life", "death", "goth", "urgently", "crisis"])

text.dispersion_plot(["repeatedly", "choice", "light", "health", "care",
"counselling", "especially"])

text.generate()

len(mini_tokens)
len(text)

len(set(text))

len(sorted(set(text)))

print(len(set(text)) / len(text))

mini_frequency = nltk.FreqDist(text)

print(mini_frequency)

mini_frequency.most_common(50)

mini_frequency.plot(25, cumulative=True)

from nltk.corpus import stopwords
nltk.download('stopwords')

stop_words = set(stopwords.words('english'))

#print(stop_words)

filter = []

for token in mini_tokens:
```

```
  token = token.lower()
  if token not in stop_words:
    filter.append(token)

#filter

filter_text = Text(filter)

len(filter_text)

len(sorted(set(filter_text)))

print((len(sorted(set(filter_text))) / len(filter_text)) * 100)

filter_frequency = nltk.FreqDist(filter_text)

filter_frequency.most_common(50)

filter_frequency.plot(25, cumulative=True)

print(type(corpus_data))

#save_tokens('cleaned5.txt', small_data)

save_tokens('corpus_word_tokens.txt', corpus_data)

#mini = sent_tokenize(mini_data)

#corpus_strings = ' '.join(corpus_words)

#corpus_strings

#corpus_words

wordcloud = WordCloud(width = 1400, height = 800, stopwords = stop_words,
min_font_size = 10).generate(str(mini_sent))

plt.figure(figsize = (12, 8), facecolor = None)
plt.imshow(wordcloud)
plt.savefig('MINI strings sent tokens 2.jpg', dpi=95, orientation='landscape')
plt.show()
```

```
"""**TOKENIZATION AND ANALYSIS OF THE TOKENS**"""

mini_sent = sent_tokenize(mini)

new_mini = [word for word in mini_sent if word.isalpha()]

new_mini

mini_sent

mini_sent_strings = [word for word in mini_sent if word.isalpha()]

mini_sent_strings

print(type(mini_sent))

#mini_sent

sent_strings = ' '.join(mini_sent)

sent_strings

corpus_tokens = word_tokenize(corpus_strings)

def save_tokens(filename, tokens):
  file_tokens = open(filename, "w")
  file_tokens.write(str(tokens))
  file_tokens.close()

tokens = word_tokenize(data)

save_tokens("mini_word_tokens.txt", mini_tokens)

tweet_tokenizer = TweetTokenizer()
mini_tweet = tweet_tokenizer.tokenize(mini_strings)

#mini_tweet

mini_tweet_strings = ' '.join(mini_tweet)

mini_tweet_strings
```

```
save_tokens("tweet_tokens.txt", tweet_tokens)

"""**REMOVE STOP WORDS**"""

nltk.download('stopwords')
from nltk.corpus import stopwords

stop_words = set(stopwords.words('english'))

document_a = word_tokenize("All of this except I don't have Netflix.
Watching this is self-harm for those of us too squeamish to watch Casualty.")

document_a

stopwords(document_a)

def stopwords(tokenized):
  filter = []
  for word in tokenized:
    if word.casefold() not in stop_words:
      filter.append(word)
  return filter

stopwords(mini_sent)

print(len(mini_sent))

stopwords(corpus_tokens)

print(type(mini_tokens))

corpus_data

corpus_list = corpus_data.split(' ')

corpus_list

mini_words = [word for word in mini_tokens if word.isalpha()]

mini_words

mini_strings = ' '.join(mini_words)
```

```
mini_strings

stopwords(tweet_tokens)

"""**FREQUENCY DISTRIBUTIONS OF EACH TOKEN**"""

def frequency(filtered_token):
  frequency = nltk.FreqDist(filtered_token)
  for key, value in frequency.items():
    print(str(key) + " : " + str(value))
  frequency.plot(30, cumulative=False)

frequency = nltk.FreqDist(mini_sent)
for key, value in frequency.items():
   print(str(key) + " : " + str(value))

plt.figure(figsize = (6, 4), facecolor = None)
frequency.plot(20, cumulative=False)
plt.savefig('sentence frequency.jpg', dpi=95, orientation='landscape')
plt.show()

plt.figure(figsize = (12, 8), facecolor = None)
plt.imshow(wordcloud)
plt.savefig('MINI strings sent tokens 2.jpg', dpi=95,
orientation='landscape')
plt.show()

print(type(frequency_image))

corpus_frequency = nltk.FreqDist(corpus_tokens)
for key, value in corpus_frequency.items():
   print(str(key) + " : " + str(value))

corpus_frequency.plot(40, cumulative=False)

filtered_tokens = stopwords(tokens)

print(type(filtered_tokens))

frequency(filtered_tokens)
```

```
filtered_word = stopwords(tokens)

frequency(filtered_word)

filtered_tweets = stopwords(tweet_tokens)

frequency(filtered_tweets)

"""**DISTANCE COMPUTATION 1 - JACCARD**"""

def jaccard_similarity(text_1, text_2):
  intersection = len(set.intersection(*[set(text_1), set(text_2)]))
  union = len(set.union(*[set(text_1), set(text_2)]))
  return intersection/float(union)

jaccard_similarity(tokens, tweet_tokens)

"""**DISTANCE COMPUTATION 2 - EUCLIDEAN** THIS WAS ABANDONED FOR NOW"""

from math import sqrt, pow, exp

def euclidean_distance(text_1, text_2):
  return sqrt(sum(pow(a-b, 2) for a, b in zip(text_1, text_2)))

#tokens_small = word_tokenize(small)

print(type(small))

from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
doc_1_vectors = vectorizer.fit_transform(tokens)

# CODE BREAKS HERE BECAUSE OF MEMORY USAGE.

doc_1_arr = doc_1_vectors.toarray()

vectorizer_2 = CountVectorizer()
doc_2_vectors = vectorizer_2.fit_transform(tweet_tokens)

#doc_2_arr = doc_2_vectors.toarray()
```

```
corpus_text = open('Twitter Suicide Corpus.txt', 'r')

corpus = ""

with open('Twitter Suicide Corpus.txt', 'r') as corpus_object:
  lines = corpus_object.readlines()
  corpus = lines

corpus = str(corpus)

print(type(corpus))

def clean_twitter_text(text):
  with open(text,"r") as read_object:
    lines = read_object.read()
    #remove special characters and punctuations
    lines = re.sub('[{}:_@\[\]0-9,%&*""?!/-]', '', lines)
    #remove the id and text tag
    lines = re.sub('(id)', '', lines)
    lines = re.sub('(text)', '', lines)
    lines = re.sub('(RT)', '', lines)
    #remove paragraph space/indentation
    lines = re.sub('(\n)', '', lines)
    return lines

corpora = clean_twitter_text('Twitter Suicide Corpora.txt')

corpora

corpora = [corpora]

corpora

print(type(corpora))

corpora

corpus = clean_twitter_text('Twitter Suicide Corpus.txt')

print(type(corpus))

roxanne = clean_twitter_text('roxanne.txt')
```

```
print(roxanne, type(roxanne))

roxanne_sent = sent_tokenize(roxanne)

roxanne_sent

corpora_sent = sent_tokenize(str(corpora))

"""**DISTANCE COMPUTATION 3 - TF-IDF**"""

from sklearn.feature_extraction.text import TfidfVectorizer

base_document = "This is an example sentence for the document to be compared"
documents = ["This is the collection of documents to be compared
against the base_document"]

vectorizer = TfidfVectorizer()
corpora.insert(0, roxanne)

embeddings = vectorizer.fit_transform(corpora)

from sklearn.metrics.pairwise import cosine_similarity, cosine_distances

similarity = cosine_similarity(embeddings[0:1], embeddings[1:]).flatten()
distance = cosine_distances(embeddings[0:1], embeddings[1:])

print(similarity)

distance

highest_score = 0
highest_score_index = 0

for i, score in enumerate(similarity):
  if highest_score < score:
    highest_score = score
    highest_score_index = i


most_similar_document = corpora[highest_score_index]
```

```
print("Most similar document by TF-IDF with the score:",
highest_score)

most_similar_document

highest_score

print(similarity.flatten())

"""**DISTANCE COMPUTATION 4 - BERT**"""

!pip install -U sentence-transformers

print(type(roxanne_sent), type(mini))

small_data = clean_twitter_text('Twitter Suicide Corpora Small.txt')

small = sent_tokenize(small_data)

print(len(small))

save_tokens("small.txt", small)

roxanne_sent

print(len(roxanne_sent))

roxanne

print(type(small))

sent_data

small.insert(0,sent_data)

from sentence_transformers import SentenceTransformer

model = SentenceTransformer('bert-base-nli-mean-tokens')

sentence_embeddings = model.encode(small)

sentence_embeddings.shape
```

```
sentence_embeddings

from sklearn.metrics.pairwise import cosine_similarity

similarity = cosine_similarity([sentence_embeddings[0]],
sentence_embeddings[1:])

save_tokens("similarity.txt", similarity)

# TO FIND THE SIMILARITY AMONG THE RANGE OF VECTORS

similars = []

for i in similarity:
  for j in i:
    similars.append(j)

print(sum(similars)/len(similars))
```

B Appendix

B.1 Creating the Document Similarity Algorithms Validation Spectrum

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import nltk

nltk.download('punkt')

base_data = pd.read_csv('medinform.csv', encoding='cp1252')

base_data['word']

frame = {
    'word': base_data['word'],
    'case_frequency': base_data['case_freq']
}

df = pd.DataFrame(frame)

case_frequency = df['case_frequency']
```

```
case_frequency.describe()

highest_frequencies = [word for word in df['case_frequency'] if word > 1567]

highest_words = []

for index, each in enumerate(df['word']):
  for index2, word in enumerate(df['case_frequency']):
    if word > 1567 and index == index2:
      highest_words.append(each)
      break

top_50 = {
    'top words': highest_words,
    'top frequencies': highest_frequencies
}

top_50_frame = pd.DataFrame(top_50)

top_50_frame

top_50_frame = top_50_frame.sort_values('top frequencies')

top_50_frame

from nltk.tokenize import TweetTokenizer, word_tokenize, sent_tokenize
import re
import os

def save_tweet_file(text, name, index):
  if text.endswith(".txt") is not True:
    with open(f'{name}_{index}.txt', 'w') as save_object:
      save_object.write(text)
      #text = f'{text[0:5]}.txt'

data = ''

def clean_twitter_text(text):
  with open(text,"r") as read_object:
    lines = read_object.read()
    #remove special characters and punctuations
    #lines = re.sub('[\n]', '', lines)
```

```
    lines = re.sub('#', '', lines)
    lines = re.sub('\\n\\n', '', lines)
    lines = re.sub('[\\n\\n]', '', lines)
    lines = re.sub('(\n\n)', '', lines)
    lines = re.sub('[{}:_@\[\]0-9,%&*""?!/-]', '', lines)
    #remove the id and text tag
    lines = re.sub('(id)', '', lines)
    lines = re.sub('(text)', '', lines)
    lines = re.sub('(RT)', '', lines)
    #remove paragraph space/indentation
    lines = re.sub('  ', '', lines)
    lines = word_tokenize(lines)
    lines = [line for line in lines if line.isalpha()]
    lines = ' '.join(lines)
  return lines

mini = clean_twitter_text('Twitter Suicide Corpora Mini.txt')

mini

mini_tokens = word_tokenize(mini)

import json

tests = []
vals = []

with open('Validation Data.txt', 'r') as val_object:
  val = val_object.read()
  val = json.loads(val)
  #print(test["data"])
  for v in val["data"]:
    vals.append(v["text"])

counter = 5

for index, value in enumerate(vals):
  while counter > 1:
    print(vals[index])
    counter -= 1
    index += 1
```

```
for index, value in enumerate(vals):
  save_tweet_file(vals[index], 'validation', index)
  print(f'done with the file number {index}')
  index += 1

print(len(vals))

all_tests = []
all_vals = []

for index in range(0, len(vals)):
  val = clean_twitter_text(f'validation_{index}.txt')
  all_vals.append(val)
  index += 1

len(all_vals)

all_vals[0]

for index, value in enumerate(all_vals):
  all_vals[index] = word_tokenize(all_vals[index])
  index += 1

all_vals[0]

import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')

stop_words = set(stopwords.words('english'))

filtered_vals = []
counter = len(all_vals)

while counter > 0:
  for i in all_vals:
    i = str(i).lower()
    if i not in stop_words:
      word_tokenize(i)
      filtered_vals.append(i)
    #print(i)
    counter -= 1
```

```
print(len(filtered_vals))

filter_mini = []

for word in mini_tokens:
  word = word.lower()
  if word not in stop_words:
    filter_mini.append(word)

filtered_vals[100]

def jaccard_similarity(text_1, text_2):
  intersection = len(set.intersection(*[set(text_1), set(text_2)]))
  union = len(set.union(*[set(text_1), set(text_2)]))
  return intersection/float(union)

import time

start = time.time()

validation_similarity_scores = []
true_positive_class = []
false_negative_class = []
no_similarity = []
threshold = 0.307435
THRESHOLD_2 = 0.28
MAX_THRESHOLD = 0.37

for index, value in enumerate(filtered_vals):
  similarity_index = jaccard_similarity(filtered_vals[index],
  filter_mini)
  similarity_index = similarity_index * 100
  similarity_index = round(similarity_index, 2)
  if similarity_index >= THRESHOLD_2:
    true_positive_class.append(similarity_index)
  if similarity_index < THRESHOLD_2:
    false_negative_class.append(similarity_index)
  if similarity_index == 0.0:
    no_similarity.append(similarity_index)
  validation_similarity_scores.append(similarity_index)
  print(f'{similarity_index}%')
```

```
  index += 1

end = time.time()

print(f'Execution time is: {(end - start) * 10**3:.03f}ms')

print(f'Max Similarity Score for the True Positive Class:
{max(true_positive_class)}')
print(f'Max Similarity Score for the False Negative Class:
{max(false_negative_class)}')
print(f'Max Similarity Score for the Validation Similarity Scores:
{max(validation_similarity_scores)}')

print(f'Minimum Similarity Score for the True Positive Class:
{min(true_positive_class)}')
print(f'Minimum Similarity Score for the False Negative Class:
{min(false_negative_class)}')
print(f'Minimum Value for the Validation Similarity Scores:
{min(validation_similarity_scores)}')

print(f'Average Similarity Score for the True Positive Class:
{(sum(true_positive_class)) / len(true_positive_class)}')
print(f'Average Similarity Score for the False Negative Class:
{(sum(false_negative_class)) / len(false_negative_class)}')
print(f'Average Similarity Score for the Validation Data:
{(sum(validation_similarity_scores)) / len(validation_similarity_scores)}')

max_true_positive_similarity_score = max(true_positive_class)
max_false_negative_similarity_score = max(false_negative_class)
max_validation_similarity_score = max(validation_similarity_scores)

min_true_positive_similarity_score = min(true_positive_class)
min_false_negative_similarity_score = min(false_negative_class)
min_validation_similarity_score = min(validation_similarity_scores)

average_true_positive_similarity_score =
(sum(true_positive_class)) / len(true_positive_class)
average_false_negative_similarity_score =
(sum(false_negative_class)) / len(false_negative_class)
average_validation_similarity_score =
(sum(validation_similarity_scores)) / len(validation_similarity_scores)
```

```
similarity_table = {
    "Maximum": [max_true_positive_similarity_score,
    max_false_negative_similarity_score, max_validation_similarity_score],
    "Minimum": [min_true_positive_similarity_score,
    min_false_negative_similarity_score, min_validation_similarity_score],
    "Average": [average_true_positive_similarity_score,
    average_false_negative_similarity_score,
    average_validation_similarity_score]
}

similarity_data_frame = pd.DataFrame(similarity_table,
index = ["True Positive Rate", "False Negative Rate", "Validation Data"])

print('SIMILARITY SCORE STATISTICS FOR THE DIFFERENT CLASSES \n')
print(similarity_data_frame)

print(len(true_positive_class))
print(len(false_negative_class))
print(len(no_similarity))
print(len(validation_similarity_scores))

print(2247/2668)

print(max(false_negative_class))

print(f'True Positive Rate: {(2247/2668) * 100}')
print(f'False Negative Rate: {(421/2668) * 100}')
```

C Appendix

C.1 Creating the Complement Spectrum

```
Note: This section has the same code as the Validation Spectrum,
I have only pasted the logic part of the code, essentially what
differentiates it from the Validation Spectrum. Full codes have
been attached and zipped.

import time

validation_similarity_scores = []
true_negative_class = []
false_positive_class = []
```

```
no_similarity = []
threshold = 0.307435
THRESHOLD_2 = 0.28
MAX_THRESHOLD = 0.37

start = time.time()

for index, value in enumerate(filtered_vals):
  similarity_index = jaccard_similarity(filtered_vals[index], filter_mini)
  similarity_index = similarity_index * 100
  similarity_index = round(similarity_index, 2)
  if similarity_index <= MAX_THRESHOLD:
    true_negative_class.append(similarity_index)
  if similarity_index > MAX_THRESHOLD:
    false_positive_class.append(similarity_index)
  if similarity_index == 0.0:
    no_similarity.append(similarity_index)
  validation_similarity_scores.append(similarity_index)
  print(f'{similarity_index}%')
  index += 1

end = time.time()

print(f'Execution Time: {(end - start)*10**3:.03f}ms')
```

D Appendix

D.1 TF-IDF and ENSEMBLE implementation

```
def tf_idf(text_1, text_2):
  vectorizer = TfidfVectorizer()
  text_2.insert(0, text_1)
  embeddings = vectorizer.fit_transform(text_2)
  cosine_similarities = cosine_similarity(embeddings[0:1],
  embeddings[1:]).flatten()
  #distance = cosine_distances(embeddings[0:1], embeddings[1:])

  highest_score = 0
  highest_score_index = 0

  for i, score in enumerate(cosine_similarities):
    if highest_score < score:
```

```
      highest_score = score
      highest_score_index = i

  most_similar_document = text_2[highest_score_index]

  #print("Most similar document by TF-IDF with the score:",
  most_similar_document, highest_score)
  return highest_score


import time

validation_similarity_scores = []
true_positive_class = []
false_negative_class = []
no_similarity = []
LOWER_PERCENTILE_THRESHOLD = 20
MEDIAN_THRESHOLD = 40
UPPER_PERCENTILE_THRESHOLD = 75

start = time.time()

for index, value in enumerate(filtered_vals):
  similarity_index_tf_idf = tf_idf(filtered_vals[index], filter_mini)
  similarity_index_tf_idf = similarity_index_tf_idf * 100
  similarity_index_tf_idf = round(similarity_index_tf_idf, 2)

  similarity_index_jaccard = jaccard_similarity(filtered_vals[index],
  filter_mini)
  similarity_index_jaccard = similarity_index_jaccard * 100
  similarity_index_jaccard = round(similarity_index_jaccard, 2)

  similarity_index = (similarity_index_tf_idf +
  similarity_index_jaccard) / 2

  if similarity_index >= MEDIAN_THRESHOLD:
    true_positive_class.append(similarity_index)
  if similarity_index < MEDIAN_THRESHOLD:
    false_negative_class.append(similarity_index)
  if similarity_index == 0.0:
    no_similarity.append(similarity_index)
  validation_similarity_scores.append(similarity_index)
```

```
  print(f'{similarity_index}%')
  index += 1

end = time.time()

print(f'Execution Time: {(end - start)*10**3:.03f}ms')
```

E Appendix

E.1 Results

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

"""Validation Spectrum - Original Data/Search Entry"""

jaccard_validation_amount_true_positives = 2247
jaccard_validation_amount_false_negatives = 421
jaccard_validation_amount_false_positives = 0
jaccard_validation_amount_true_negatives = 0
jaccard_validation_amount_experiments = 2668
jaccard_validation_true_positive_rate = round(((jaccard_validation_
amount_true_positives / jaccard_validation_
amount_experiments) * 100), 3)
jaccard_validation_false_negative_rate = round(((jaccard_validation_
amount_false_negatives
/ jaccard_validation_amount_experiments) * 100), 3)
jaccard_recall = jaccard_validation_true_positive_rate
/ (jaccard_validation_true_positive_rate + jaccard_validation_
false_negative_rate)
jaccard_validation_accuracy_score = round(((
(jaccard_validation_amount_true_positives + jaccard_validation_
amount_true_negatives)
/ jaccard_validation_amount_experiments) * 100), 3)
jaccard_validation_computation_time = 23603.915
jaccard_validation_error_rate = round((1 - jaccard_validation_
accuracy_score), 3)
jaccard_threshold = 0.28

"""Similar Spectrum"""

jaccard_similar_amount_true_positives = 3760
```

```
jaccard_similar_amount_false_negatives = 10
jaccard_similar_amount_false_positives = 0
jaccard_similar_amount_true_negatives = 0
jaccard_similar_amount_experiments = 3770
jaccard_similar_true_positive_rate = round(((jaccard_similar_amount_true_positives / jaccard
jaccard_similar_false_negative_rate = round(((jaccard_similar_amount_false_negatives
/ jaccard_similar_amount_experiments) * 100), 3)
jaccard_similar_recall = jaccard_similar_true_positive_rate /
(jaccard_similar_true_positive_rate +
jaccard_similar_false_negative_rate)
jaccard_similar_accuracy_score = round((((jaccard_similar_amount_true_positives + jaccard_si
jaccard_similar_amount_experiments) * 100), 3)
jaccard_similar_computation_time = 28002.359
jaccard_similar_error_rate = round((1 - jaccard_similar_accuracy_score), 3)
jaccard_similar_threshold = 0.37

"""Complement Spectrum"""

jaccard_complement_amount_true_positives = 0
jaccard_complement_amount_false_negatives = 0
jaccard_complement_amount_false_positives = 215
jaccard_complement_amount_true_negatives = 3252
jaccard_complement_amount_experiments = 3467
jaccard_complement_true_negative_rate = round(((jaccard_complement
_amount_true_negatives / jaccard_complement_amount_experiments) * 100), 3)
jaccard_complement_false_positive_rate = round(((jaccard_complement
_amount_false_positives / jaccard_complement_amount_experiments) * 100), 3)
jaccard_complement_precision = jaccard_complement_amount_true_positives / (jaccard_complemen
_amount_false_positives)
jaccard_complement_accuracy_score = round((((jaccard_complement_
amount_true_positives + jaccard_complement_amount_true_negatives)
/ jaccard_complement_amount_experiments) * 100), 3)
jaccard_complement_computation_time = 25955.301
jaccard_complement_error_rate = round((1 - jaccard_complement_accuracy_score), 3)
jaccard_complement_threshold = 0.37

"""Validation - TF-IDF"""

tf_idf_validation_amount_true_positives = 1449
tf_idf_validation_amount_false_negatives = 1219
tf_idf_validation_amount_false_positives = 0
tf_idf_validation_amount_true_negatives = 0
```

```
tf_idf_validation_amount_experiments = 2668
tf_idf_validation_true_positive_rate = round(((tf_idf_validation_amount_
true_positives / tf_idf_validation_amount_experiments) * 100), 3)
tf_idf_validation_false_negative_rate = round(((tf_idf_validation_
amount_false_negatives / tf_idf_validation_amount_experiments) * 100), 3)
tf_idf_validation_recall = tf_idf_validation_true_positive_rate /
(tf_idf_validation_true_positive_rate + tf_idf_validation_false_negative_rate)
tf_idf_validation_accuracy_score = round((((tf_idf_
validation_amount_true_positives + tf_idf_validation_amount_true_negatives) /
tf_idf_validation_amount_experiments) * 100), 3)
tf_idf_validation_computation_time = 616172.385
tf_idf_validation_error_rate = round((1 - tf_idf_validation_accuracy_score), 3)
tf_idf_validation_threshold = 50

"""Validation - Similar Spectrum - TF-IDF"""

tf_idf_similar_amount_true_positives = 1779
tf_idf_similar_amount_false_negatives = 1991
tf_idf_similar_amount_false_positives = 0
tf_idf_similar_amount_true_negatives = 0
tf_idf_similar_amount_experiments = 3770
tf_idf_similar_true_positive_rate = round(((tf_idf_similar_
amount_true_positives
/ tf_idf_similar_amount_experiments) * 100), 3)
tf_idf_similar_false_negative_rate = round(((tf_idf_similar_
amount_false_negatives / tf_idf_similar_amount_experiments) * 100), 3)
tf_idf_similar_recall = round(tf_idf_similar_amount_true_positives /
(tf_idf_similar_amount_true_positives + tf_idf_similar
_amount_false_negatives), 3)
tf_idf_similar_accuracy_score = round((((tf_idf_similar
_amount_true_positives + tf_idf_similar_amount_true_negatives)
/ tf_idf_similar_amount_experiments) * 100), 3)
tf_idf_similar_computation_time = 1010864.712
tf_idf_similar_error_rate = round((1 - tf_idf_similar_accuracy_score), 3)
tf_idf_similar_threshold = 50

print(tf_idf_similar_true_positive_rate, tf_idf_similar_false_negative_rate,
tf_idf_similar_recall, tf_idf_similar_accuracy_score)

"""Validation - Complement Spectrum - TF-IDF"""

tf_idf_complement_amount_true_positives = 0
```

```
tf_idf_complement_amount_false_negatives = 0
tf_idf_complement_amount_false_positives = 1433
tf_idf_complement_amount_true_negatives = 2034
tf_idf_complement_amount_experiments = 3467
tf_idf_complement_true_negative_rate = round(((tf_idf_complement_
amount_true_negatives / tf_idf_complement_amount_experiments) * 100), 3)
tf_idf_complement_false_positive_rate = round(((tf_idf_complement_
amount_false_positives / tf_idf_complement_amount_experiments) * 100), 3)
tf_idf_complement_precision = tf_idf_complement_amount_true_positives
/ (tf_idf_complement_amount_true_positives + tf_idf_complement_amount_
false_positives)
tf_idf_complement_accuracy_score = round((((tf_idf_complement_
amount_true_positives + tf_idf_complement_amount_true_negatives)
/ tf_idf_complement_amount_experiments) * 100), 3)
tf_idf_complement_computation_time = 935262.446
tf_idf_complement_error_rate = round((1 - tf_idf_
complement_accuracy_score), 3)
tf_idf_complement_threshold = 50

"""Validation - Ensemble"""

ensemble_validation_amount_true_positives = 1370
ensemble_validation_amount_false_negatives = 1298
ensemble_validation_amount_false_positives = 0
ensemble_validation_amount_true_negatives = 0
ensemble_validation_amount_experiments = 2668
ensemble_validation_true_positive_rate = round(((ensemble_
validation_amount_true_positives / ensemble_
validation_amount_experiments) * 100), 3)
ensemble_validation_false_negative_rate = round(((ensemble_
validation_amount_false_negatives / ensemble_validation_
amount_experiments) * 100), 3)
ensemble_validation_recall = ensemble_validation_amount_true_positives / (ensemble_validatio
validation_amount_false_positives)
ensemble_validation_accuracy_score = round((((ensemble_
validation_amount_true_positives + ensemble_validation_amount_true_negatives) / ensemble_val
ensemble_validation_computation_time = 640956.478
ensemble_validation_error_rate = round((1 -
ensemble_validation_accuracy_score), 3)
ensemble_validation_threshold = 40

"""Validation - Similar Spectrum - Ensemble"""
```

```
ensemble_similar_amount_true_positives = 1553
ensemble_similar_amount_false_negatives = 2217
ensemble_similar_amount_false_positives = 0
ensemble_similar_amount_true_negatives = 0
ensemble_similar_amount_experiments = 3770
ensemble_similar_true_positive_rate = round((
(ensemble_similar_amount_true_positives / ensemble_similar_amount_experiments)
* 100), 3)
ensemble_similar_false_negative_rate = round(((ensemble_similar_
amount_false_negatives / ensemble_similar_amount_experiments) * 100), 3)
ensemble_similar_recall = ensemble_similar_amount_true_positives /
(ensemble_similar_amount_true_positives + ensemble_similar_
amount_false_positives)
ensemble_similar_accuracy_score = round((((ensemble_similar_
amount_true_positives + ensemble_similar_amount_true_negatives)
/ ensemble_similar_amount_experiments) * 100), 3)
ensemble_similar_computation_time = 827789.851
ensemble_similar_error_rate = round((1 - ensemble_similar_accuracy_score), 3)
ensemble_similar_threshold = 40

"""Validation - Complementary Spectrum - Ensemble"""

ensemble_complement_amount_true_positives = 0
ensemble_complement_amount_false_negatives = 0
ensemble_complement_amount_false_positives = 0
ensemble_complement_amount_true_negatives = 3467
ensemble_complement_amount_experiments = 3467
ensemble_complement_true_negative_rate = round(((ensemble_complement_
amount_true_negatives / ensemble_complement_amount_experiments) * 100), 3)
ensemble_complement_false_positive_rate = round(((ensemble_
complement_amount_false_positives / ensemble_complement_amount
_experiments) * 100), 3)
ensemble_complement_precision = ensemble_complement_amount
_true_positives / (ensemble_complement_amount_experiments)
ensemble_complement_accuracy_score = round((((ensemble_
complement_amount_true_positives + ensemble_complement_amount_true_negatives) / ensemble_com
ensemble_complement_computation_time = 1690820.646
ensemble_complement_error_rate = round((1 - ensemble_
complement_accuracy_score), 3)
ensemble_complement_threshold = 55
```

```
"""Validation - BERT"""

bert_validation_amount_true_positives = 11
bert_validation_amount_false_negatives = 1
bert_validation_amount_false_positives = 0
bert_validation_amount_true_negatives = 0
bert_validation_amount_experiments = 12
bert_validation_true_positive_rate = round(((bert_validation_amount
_true_positives / bert_validation_amount_experiments) * 100), 3)
bert_validation_false_negative_rate = round(((bert_validation_amount
_false_negatives / bert_validation_amount_experiments) * 100), 3)
bert_validation_recall = bert_validation_amount_true_positives /
(bert_validation_amount_true_positives + bert_validation_amount_false_positives)
bert_validation_accuracy_score = round((((bert_validation_amount_true_positives + bert_valid
bert_validation_amount_experiments) * 100), 3)
bert_validation_computation_time = 0 #10800000
bert_validation_error_rate = round((1 - bert_validation_accuracy_score), 3)
bert_validation_prevalence_rate = round((bert_validation_amount_true_positives/ bert_validat
bert_validation_threshold = 30

"""Validation - Similar Spectrum - BERT"""

bert_similar_amount_true_positives = 25
bert_similar_amount_false_negatives = 0
bert_similar_amount_false_positives = 0
bert_similar_amount_true_negatives = 0
bert_similar_amount_experiments = 25
bert_similar_true_positive_rate = round((
(bert_similar_amount_true_positives /
bert_similar_amount_experiments) * 100), 3)
bert_similar_false_negative_rate = round(((bert_similar_
amount_false_negatives / bert_similar_amount_experiments) * 100), 3)
bert_similar_recall = bert_similar_amount_true_positives /
(bert_similar_amount_true_positives + bert_similar_amount_false_positives)
bert_similar_accuracy_score = round((((bert_similar_
amount_true_positives + bert_similar_amount_true_negatives) /
bert_similar_amount_experiments) * 100), 3)
bert_similar_computation_time = 21600000
bert_similar_threshold = 30

"""Validation - Complementary Spectrum - BERT"""
```

```
bert_complement_amount_true_positives = 0
bert_complement_amount_false_negatives = 0
bert_complement_amount_false_positives = 0
bert_complement_amount_true_negatives = 10
bert_complement_amount_experiments = 10
bert_complement_true_negative_rate = round(((bert_complement_
amount_true_negatives / bert_complement_amount_experiments) * 100), 3)
bert_complement_false_positive_rate = round(((bert_complement_
amount_false_positives / bert_complement_amount_experiments) * 100), 3)
bert_complement_precision = bert_complement_amount_true_positives /
(bert_complement_amount_experiments)
bert_complement_accuracy_score = round(((
(bert_complement_amount_true_positives +
bert_complement_amount_true_negatives) /
bert_complement_amount_experiments) * 100), 3)
bert_complement_computation_time = 15480000
bert_complement_threshold = 30

results_validation = {
    "Jaccard": [jaccard_validation_true_positive_rate,
    jaccard_validation_false_negative_rate, jaccard_recall,
    jaccard_validation_accuracy_score, jaccard_validation
    _amount_experiments, jaccard_validation_computation_time,
    jaccard_threshold],
    "TF-IDF": [tf_idf_validation_true_positive_rate,
    tf_idf_validation_false_negative_rate, tf_idf_validation_recall,
    tf_idf_validation_accuracy_score, tf_idf_validation_amount_experiments,
    tf_idf_validation_computation_time, tf_idf_validation_threshold],
    "ENSEMBLE": [ensemble_validation_true_positive_rate, ensemble_validation
    _false_negative_rate, ensemble_validation_recall, ensemble_
    validation_accuracy_score, ensemble_validation_amount_experiments,
    ensemble_validation_computation_time, ensemble_validation_threshold],
    "BERT*": [bert_validation_true_positive_rate, bert_validation_false_
    negative_rate, bert_validation_recall, bert_validation_accuracy_score,
    bert_validation_amount_experiments, f'{bert_validation_computation_time}*',
    bert_validation_threshold]
}

results_data_frame_validation = pd.DataFrame(results_validation,
index = ["True Positive Rate", "False Negative Rate", "Recall",
"Accuracy Score", "Experiments Count", "Computation Time (ms)",
"Threshold"])
```

```
print('VALIDATION SPECTRUM WITH THE MEDIAN THRESHOLD \n')
print(results_data_frame_validation)

compute = {
    "Jaccard": 23603.9150,
    "TF-IDF": 616172.3850,
    "Ensemble": 640956.478,
    "BERT": 11520000
}

compute_df = pd.DataFrame(compute, index=[1, 2, 3, 4])

models_name = ["Jaccard", "TF-IDF", "Ensemble", "BERT"]
computation_time = [23603.9150, 616172.3850, 640956.478, 11520000]

experiments_count = [2668, 2668, 2668, 12]

plt.barh(models_name, computation_time, color='violet')
plt.xlabel("Computation Time (ms)")
plt.ylabel("Model Names")
#plt.title("Computation Time of the Different Models")
#plt.savefig('Computation Time of the Different Models.jpg',
dpi=95, orientation='landscape')
plt.show()

plt.bar(models_name, experiments_count, color='violet')
plt.xlabel("Model Names")
plt.ylabel("Experiments Count")
#plt.title("Computation Time of the Different Models")
#plt.savefig('Experiment Count of the Different Models -
Validation Spectrum.jpg', dpi=95, orientation='landscape')
plt.show()

results_similar = {
    "Jaccard": [jaccard_similar_true_positive_rate,
    jaccard_similar_false_negative_rate, jaccard_similar_recall,
    jaccard_similar_accuracy_score, jaccard_similar_amount_experiments,
    jaccard_similar_computation_time, jaccard_similar_threshold],
    "TF-IDF": [tf_idf_similar_true_positive_rate, tf_idf_similar_
    false_negative_rate, tf_idf_similar_recall, tf_idf_similar_accuracy_score,
    tf_idf_similar_amount_experiments, tf_idf_similar_computation_time,
```

```
    tf_idf_similar_threshold],
    "ENSEMBLE": [ensemble_similar_true_positive_rate, ensemble_
    similar_false_negative_rate, ensemble_similar_recall, ensemble_
    similar_accuracy_score, ensemble_similar_amount_experiments,
    ensemble_similar_computation_time, ensemble_similar_threshold],
    "BERT*": [bert_similar_true_positive_rate, bert_similar_false_
    negative_rate, bert_similar_recall, bert_similar_accuracy_score,
    bert_similar_amount_experiments, bert_similar_computation_time,
    bert_similar_threshold]
}

results_data_frame_similar = pd.DataFrame(results_similar,
index = ["True Positive Rate", "False Negative Rate", "Recall",
"Accuracy Score", "Experiments Count", "Computation Time (ms)",
"Threshold"])

print('VALIDATION - SIMILAR SPECTRUM WITH THE MEDIAN THRESHOLD \n')
print(results_data_frame_similar)

tpr_rates = [results_data_frame_similar["ENSEMBLE"]
["True Positive Rate"], results_data_frame_similar["TF-IDF"]
["True Positive Rate"],
results_data_frame_similar["Jaccard"]["True Positive Rate"],
results_data_frame_similar["BERT*"]["True Positive Rate"]]
tpr_names = ["ENSEMBLE", "TF-IDF", "Jaccard", "BERT"]

plt.bar(tpr_names, tpr_rates, color='violet')
plt.xlabel("Model Names")
plt.ylabel("True positive Rates")
#plt.title("Computation Time of the Different Models")
plt.savefig('True Positive Rates of the Different Models -
Similar Spectrum.jpg', dpi=95, orientation='landscape')
plt.show()

fnr_rates = [results_data_frame_similar["BERT*"]
["False Negative Rate"], results_data_frame_similar["Jaccard"]
["False Negative Rate"],
results_data_frame_similar["TF-IDF"]["False Negative Rate"],
results_data_frame_similar["ENSEMBLE"]["False Negative Rate"]]
fnr_names = ["BERT", "Jaccard", "TF-IDF", "ENSEMBLE"]

plt.bar(fnr_names, fnr_rates, color='violet')
```

```
plt.xlabel("Model Names")
plt.ylabel("False Negative Rates")
#plt.title("Computation Time of the Different Models")
plt.savefig('False Negative Rates of the Different Models -
Similar Spectrum.jpg', dpi=95, orientation='landscape')
plt.show()

recall_rates = [results_data_frame_similar["TF-IDF"]["Recall"],
results_data_frame_similar["Jaccard"]["Recall"],
results_data_frame_similar["ENSEMBLE"]["Recall"],
results_data_frame_similar["BERT*"]["Recall"]]
recall_names = ["TF-IDF", "Jaccard", "ENSEMBLE", "BERT"]

plt.bar(recall_names, recall_rates, color='violet')
plt.xlabel("Model Names")
plt.ylabel("Recall")
#plt.title("Computation Time of the Different Models")
plt.savefig('Recall Rates of the Different Models -
Similar Spectrum.jpg', dpi=95, orientation='landscape')
plt.show()

accuracy_rates = [results_data_frame_similar["ENSEMBLE"]
["True Positive Rate"], results_data_frame_similar["TF-IDF"]
["True Positive Rate"],
results_data_frame_similar["Jaccard"]["True Positive Rate"],
results_data_frame_similar["BERT*"]["True Positive Rate"]]
accuracy_names = ["ENSEMBLE", "TF-IDF", "Jaccard", "BERT"]

plt.bar(accuracy_names, accuracy_rates, color='violet')
plt.xlabel("Model Names")
plt.ylabel("Accuracy Rates (%)")
#plt.title("Computation Time of the Different Models")
plt.savefig('Accuracy Rates of the Different Models - Similar
Spectrum.jpg', dpi=95, orientation='landscape')
plt.show()

results_complement = {
    "Jaccard": [jaccard_complement_true_negative_rate,
    jaccard_complement_false_positive_rate, jaccard_
    complement_accuracy_score, jaccard_complement_amount_experiments,
    jaccard_complement_computation_time, jaccard_complement_threshold],
    "TF-IDF": [tf_idf_complement_true_negative_rate, tf_idf_
```

```
    complement_false_positive_rate, tf_idf_complement_accuracy_
    score, tf_idf_complement_amount_experiments, tf_idf_similar_
    computation_time, tf_idf_complement_threshold],
    "ENSEMBLE": [ensemble_complement_true_negative_rate, ensemble_
    complement_false_positive_rate, ensemble_complement_accuracy_
    score, ensemble_complement_amount_experiments, ensemble_
    complement_computation_time, ensemble_complement_threshold],
    "BERT*": [bert_complement_true_negative_rate, bert_complement
    _false_positive_rate, bert_complement_accuracy_score,
    bert_complement_amount_experiments, bert_complement_computation_time,
    bert_complement_threshold]
}

results_data_frame_complement = pd.DataFrame(results_complement,
index = ["True Negative Rate", "False Positive Rate",
"Accuracy Score", "Experiments Count", "Computation Time (ms)",
"Threshold"])

print('VALIDATION - COMPLEMENT SPECTRUM WITH THE MEDIAN THRESHOLD \n')
print(results_data_frame_complement)

tnr_rates = [results_data_frame_complement["TF-IDF"]
["True Negative Rate"], results_data_frame_complement["Jaccard"]
["True Negative Rate"],
results_data_frame_complement["ENSEMBLE"]["True Negative Rate"],
results_data_frame_complement["BERT*"]["True Negative Rate"]]
tnr_names = ["TF-IDF", "Jaccard", "ENSEMBLE", "BERT"]

plt.bar(tnr_names, tnr_rates, color='violet')
plt.xlabel("Model Names")
plt.ylabel("True Negative Rates")
#plt.title("Computation Time of the Different Models")
plt.savefig('True Negative Rates of the Different Models -
Complement Spectrum.jpg', dpi=95, orientation='landscape')
plt.show()

fpr_rates = [results_data_frame_complement["ENSEMBLE"]
["False Positive Rate"], results_data_frame_complement["BERT*"]
["False Positive Rate"],
results_data_frame_complement["Jaccard"]["False Positive Rate"],
results_data_frame_complement["TF-IDF"]["False Positive Rate"]]
fpr_names = ["ENSEMBLE", "BERT", "Jaccard", "TF-IDF"]
```

```
plt.bar(fpr_names, fpr_rates, color='violet')
plt.xlabel("Model Names")
plt.ylabel("False Positive Rates")
#plt.title("Computation Time of the Different Models")
plt.savefig('False Positive Rates of the Different Models -
Complement Spectrum.jpg', dpi=95, orientation='landscape')
plt.show()

accuracy_complement_rates = [results_data_frame_complement["TF-IDF"]
["Accuracy Score"], results_data_frame_complement["Jaccard"]
["Accuracy Score"], results_data_frame_complement["ENSEMBLE"]
["Accuracy Score"], results_data_frame_complement["BERT*"]
["Accuracy Score"]]
accuracy_complement_names = [ "TF-IDF", "Jaccard",
"ENSEMBLE", "BERT"]

plt.bar(accuracy_complement_names, accuracy_complement_rates,
color='violet')
plt.xlabel("Model Names")
plt.ylabel("Accuracy Rates (%)")
#plt.title("Computation Time of the Different Models")
plt.savefig('Accuracy Rates of the Different Models -
Complement Spectrum.jpg', dpi=95, orientation='landscape')
plt.show()
```

F Appendix

F.1 SSL

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import nltk
from nltk.tokenize import TweetTokenizer, word_tokenize, sent_tokenize
import re
import os
nltk.download('punkt')

data = ''

def clean_twitter_text(text):
  with open(text,"r") as read_object:
```

```
    lines = read_object.read()
    #remove special characters and punctuations
    #lines = re.sub('[\n]', '', lines)
    lines = re.sub('#', '', lines)
    lines = re.sub('\\n\\n', '', lines)
    lines = re.sub('[\\n\\n]', '', lines)
    lines = re.sub('(\n\n)', '', lines)
    lines = re.sub('[{}:_@\[\]0-9,%&*""?!/-]', '', lines)
    #remove the id and text tag
    lines = re.sub('(id)', '', lines)
    lines = re.sub('(text)', '', lines)
    lines = re.sub('(RT)', '', lines)
    #remove paragraph space/indentation
    lines = re.sub('  ', '', lines)
    lines = word_tokenize(lines)
    lines = [line for line in lines if line.isalpha()]
    lines = ' '.join(lines)
  return lines

self = clean_twitter_text('self harm tweet search - 1000.txt')

self

print(type(self))

from nltk.corpus import stopwords
nltk.download('stopwords')

self_tokens = word_tokenize(self)

self_tokens

stop_words = set(stopwords.words('english'))

filter_self = []

for word in self_tokens:
  word = word.lower()
  if word not in stop_words:
    filter_self.append(word)

len(filter_self)
```

```
filter_self

len(set(filter_self))

filter_unique = set(filter_self)

len(filter_unique)

filter_text = nltk.Text(filter_self)

filter_frequency = nltk.FreqDist(filter_self)

filter_frequency.plot(25, cumulative = True)

filter_text.dispersion_plot(['self', 'someone', 'sex', 'trust', 'harm',
'communicate', 'hoodhealer', 'selfharm', 'mental', 'health', 'care',
'light', 'counselling', 'manasra', 'especially', 'must', 'necessary'])

filter_frequency.most_common(20)

filter_frequency.tabulate()

filter_text.collocation_list()

from wordcloud import WordCloud, STOPWORDS

stopwords = set(STOPWORDS)

clean = ''

for i in filter_self:
  i = str(i)
  tokens = i.split()
  for i in range(len(tokens)):
    tokens[i] = tokens[i].lower()
  clean += " ".join(tokens)+" "
wordcloud = WordCloud(width = 1400, height = 800, stopwords = stopwords,
min_font_size = 10).generate(clean)

plt.figure(figsize = (12, 8), facecolor = None)
plt.imshow(wordcloud)
```

```
plt.savefig('SSL WordCloud.jpg', dpi=95, orientation='landscape')
plt.show()

len(filter_frequency)

self_text = nltk.Text(self_tokens)

self_text

self_text.generate()

self_text.concordance('harm')
```

References

Alsentzer, E., Murphy, J. R., Boag, W., Weng, W.-H., Jin, D., Naumann, T. & McDermott, M. (2019), 'Publicly available clinical bert embeddings', *arXiv preprint arXiv:1904.03323* .

Bidargaddi, N., Musiat, P., Makinen, V.-P., Ermes, M., Schrader, G. & Licinio, J. (2017), 'Digital footprints: facilitating large-scale environmental psychiatric research in naturalistic settings through data from everyday technologies', *Molecular psychiatry* **22**(2), 164–169.

Boers, E., Afzali, M. H., Newton, N. & Conrod, P. (2019), 'Association of screen time and depression in adolescence', *JAMA pediatrics* **173**(9), 853–859.

Brevard, A., Lester, D. & Yang, B. (1990), 'A comparison of suicide notes written by suicide completers and suicide attempters.', *Crisis: The Journal of Crisis Intervention and Suicide Prevention* .

Clayton, R. & Clayton, C. (2022), 'Uk screen use in 2022: A need for guidance'.

Coppersmith, G., Leary, R., Crutchley, P. & Fine, A. (2018), 'Natural language processing of social media as screening for suicide risk', *Biomedical informatics insights* **10**, 1178222618792860.

Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. (2018), 'Bert: Pre-training of deep bidirectional transformers for language understanding', *arXiv preprint arXiv:1810.04805* .

Gerlach, M., Shi, H. & Amaral, L. A. N. (2019), 'A universal information theoretic approach to the identification of stopwords', *Nature Machine Intelligence* **1**(12), 606–612.

Hall, A. (2018), 'Average briton gets six hours and 19 minutes of sleep a night, study finds'.

Haque, A., Reddi, V. & Giallanza, T. (2021), Deep learning for suicide and depression identification with unsupervised label correction, *in* 'International Conference on Artificial Neural Networks', Springer, pp. 436–447.

Horev, R. (2018), 'Bert explained: State of the art language model for nlp'.
URL: *https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270*

Josh Gerstein, A. W. (2022), 'Supreme court has voted to overturn abortion rights, draft opinion shows'.
URL: *https://www.politico.com/news/2022/05/02/supreme-court-abortion-draft-opinion-00029473*

Klonsky, E. D. & May, A. M. (2015), 'The three-step theory (3st): A new theory of suicide rooted in the "ideation-to-action" framework', *International Journal of Cognitive Therapy* **8**(2), 114–129.

Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C. H. & Kang, J. (2020), 'Biobert: a pre-trained biomedical language representation model for biomedical text mining', *Bioinformatics* **36**(4), 1234–1240.

Luxton, D. D., June, J. D. & Fairall, J. M. (2012), 'Social media and suicide: a public health perspective', *American journal of public health* **102**(S2), S195–S200.

Mikheev, A. (2003), Text segmentation, *in* 'The Oxford handbook of computational linguistics', Oxford Handbook of Computational Linguistics.

Miller, G. A., Leacock, C., Tengi, R. & Bunker, R. T. (1993), A semantic concordance, *in* 'Human Language Technology: Proceedings of a Workshop Held at Plainsboro, New Jersey, March 21-24, 1993'.

Mohammad, S. M. (2017), 'Word affect intensities', *arXiv preprint arXiv:1704.08798* .

Pennington, J., Socher, R. & Manning, C. D. (2014), Glove: Global vectors for word representation, *in* 'Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)', pp. 1532–1543.

Rawat, B. P. S., Kovaly, S., Pigeon, W. R. & Yu, H. (2022), 'Scan: Suicide attempt and ideation events dataset', *arXiv preprint arXiv:2205.07872* .

Roubidoux, S. M. (2012), Linguistic manifestations of power in suicide notes: An investigation of personal pronouns, PhD thesis.

samaritan.org (2020), 'Latest suicide data'.
URL: *https://www.samaritans.org/about-samaritans/research-policy/suicide-facts-and-figures/latest-suicide-data/*

Scott, W. (2019), 'Tf-idf from scratch in python on a real-world dataset'.
URL: *https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-dataset-796d339a4089*

Shao, Z., Chandramouli, R., Subbalakshmi, K. & Boyadjiev, C. T. (2019), 'An analytical system for user emotion extraction, mental state modeling, and rating', *Expert Systems with Applications* **124**, 82–96.

Solanke, S. O., AMOLE, F. & Onugu, P. (2022), 'Language, socio-cultural symbolisms and implications in yoruba hieroglyphics', *Issues in Language and Literary Studies* **5**(1).

Wang, N., Luo, F., Shivtare, Y., Badal, V. D., Subbalakshmi, K., Chandramouli, R. & Lee, E. (2021), 'Learning models for suicide prediction from social media posts', *arXiv preprint arXiv:2105.03315* .

who.int (2021), 'Suicide'.
URL: *https://www.who.int/news-room/fact-sheets/detail/suicide*