

PROFESSIONAL TRAINING REPORT
at
Sathyabama Institute of Science and Technology
(Deemed to be University)

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering Degree in Computer Science and Engineering

By

M DHARSHINI NITHILA
(38110127)

and

VINISHA R
(38110646)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING
SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI – 600119, TAMILNADU

MARCH

2022



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with Grade “A” by NAAC

(Established under Section 3 of UGC Act, 1956)

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI– 600119

www.sathyabamauniversity.ac.in



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **M DHARSHINI NITHILA (38110127)** and **VINISHA R (38110127)** who carried out the project entitled “**Predicting Harmful Web Pages Based on Suicidal- related Textual Analysis Using Machine Learning Algorithms**” under my supervision from June 2021 to Nov 2021.

Internal Guide

Mr. A.Yovan Felix M.E.,Ph.D.,

Head of the Department

Dr. L. Lakshmanan, M.E., Ph.D.,

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I M DHARSHINI NITHILA and VINISHA R hereby declare that the Project Report entitled **Predicting Harmful Web Pages Based on Suicidal- related Textual Analysis Using Machine Learning Algorithms** done by me under the guidance of **Mr.A.Yovan Felix, M.E.,Ph.D.**, is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE: 01/03/2022

M DHARSHINI NITHILA and VINISHA R

PLACE: CHENNAI

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management** of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T.Sasikala M.E., Ph.D., Dean**, School of Computing , **Dr.S.Vigneshwari M.E., Ph.D., and Dr.L.Lakshmanan M.E., Ph.D.,** Heads of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Mr.A.Yovan Felix, M.E.,Ph.D.,** for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

Due to multiple reasons like the nuclear family, peer pressure for fake prestige, impatience attitude, and mental stress has become a common trait in every person. With advancements in technology like the internet and online presence, it has become a routine to be active online. Some sections of people vent out their emotions online as they have no support system in real life. It has been detected, as seen in some instances; those suicidal tendencies ranging from mild to extreme could be from a person's online profile activity. In our current work, we use a specific method that includes all critical criteria that could be exhibited by a suicidal person by using Natural Language Processing (NLP) techniques. NLP interprets written language, whereas Machine Learning makes predictions based on patterns learned from experience. These textual features are passed through a robust Machine Learning framework for detecting an abrupt change in input data. Our method predicts efficiently a genuine, mentally disturbed profile from a typical profile.

	TABLE OF CONTENTS	
CHAPTER No.	TITLE	PAGE No
	ABSTRACT	I
	LIST OF FIGURES	II
	LIST OF TABLES	III
1	INTRODUCTION	1
2	LITERATURE SURVEY	5
3	METHODOLOGY	8
4	RESULTS AND DISCUSSION, PERFORMANCE ANALYSIS	12
5	FUTURE WORK AND CONCLUSIONS	14
	REFERENCES	15
	APPENDIX	16
	A.SCREENSHOTS	16
	B. SOURCE CODE	17

LIST OF FIGURES

FIGURE NO.	FIGURE NAME	PAGE NO
1	Prediction graph	05
2	Points and Line	06
3	Clustered data	07
4	Original Unclustered data	16
5	Select the Url	17
6	Detects without Suicidal content the url	17
7	Select the Url	17
8	Detects without Suicidal content the url	17

LIST OF TABLES

TABLE NO.	TABLE NAME	PAGE NO
1	Prediction Table	17

CHAPTER 1

INTRODUCTION

At the moment, the task of preventing suicide is relevant, because according to statistics of the World Health Organization every year more than 800 000 people commit suicide. According to the same data, the Russian Federation is among the five countries with the highest number of suicides per 100,000 inhabitants. Since the Internet is the easiest way to distribute suicidal content in the form of various web pages dedicated to suicide, a large number of organizations are trying to solve this problem, for example, the popular social network Facebook, which detects suspicious profiles of users prone to suicide, and posts containing suicidal content. In addition to social networks, federal services are trying to solve the problem, for example, in Russia Roskomnadzor in 2016 published recommendations for disseminating information about suicide cases in the media, which probably affects the results of search engines on this topic. In addition, since 2006 there has been a unified register that contains websites blocked in the Russian Federation. However, blocking does not happen immediately. Some people manage to visit dangerous web pages. Manual blocking of suicidal websites can hardly be called an effective measure in the fight against the spread of suicidal content. Since after several years of such blockages, the number of child suicides in Russia has risen sharply again. In addition to the «death groups» in social networks that platform developers are already actively fighting, one of the reasons may be that websites with suicidal content can create their own copies (mirrors). This article discusses the possibility of detecting such web pages by analyzing their content in real time using machine learning algorithms. Detection occurs on the client's side. In this way, with sufficient accuracy to identify dangerous websites visited by the user, it is possible to identify a person who is suicidal at an early stage.

CHAPTER 2

LITERATURE SURVEY

The SGD algorithm is good at identifying harmful websites, but it can make mistakes when evaluating which ones are safe, labelling them as risky.[1]The web mining algorithm will extract textual information from web pages and identify those associated with terrorism. A system whose main purpose is to create a website where people may inspect any webpage or website for any evidence of terrorist activity.[2]The persuading concept is to see if the feature's equivalent word appears in the mail. When a good classifier is employed to create the classification model, the experimental results of this method have high TPR and Precision values, and the false positive rate is regulated within an acceptable range.[3]Linguistic characteristics are critical for distinguishing across users' written styles. Sentiment analysis is most effective with content that has a subjective context, such as a suicide note. These characteristics can be derived explicitly from the user profile or inferred implicitly using various data mining tools and methodologies.[4]Using a document embedding, A decision tree model with gradient boosting predicts dangerous categories of gathered web pages. [5] The ROC curve was used to comprehend a performance measurement for a classification task at various thresholds. The false-positive percentage should be kept as low as possible, whereas the true positive rate should be maximized. [6]Hierarchical spatial scaling's analytic bias improves the model's ability to handle detection problems in documents of possibly changing sizes. A feature extractor is a programme that parses a neural network model that distinguishes a series of tokens from HTML pages judgments, in this approach.[7]Cross-channel scripting defence methods follow a website's whole path, consisting of sustained storage systems If the soiled information is not cleansed, an alert is generated. The adversary can use this threat to insert inappropriate material into the user's embedded system. causing web applications to malfunction and information to be leaked.[8]Clustering methods lustring (e.g., k-means, DBSCAN) To detect malicious domains, unsupervised separation instances

evaluate data and derive necessary details from it.[9]The Dirichlet latent allocation topic model proposed a pattern that determines that tweets about crime are more likely to be positive. However, A stash of tweets is available; however, a series of old tweets is either impossible or prohibitively expensive.

CHAPTER 3

METHODOLOGY

We propose a system with the primary goal of creating a website where users can search for any trace of terrorist activity on any web page or website. To accomplish this, our website will allow users to enter the URL of the web page they wish to scan. After entering the URL, our system will count the words on the entire web page and compare them to the words already in our database. Each word that we store in our database will be assigned a score. Our system will retrieve the scores for each word on the user's web page from our database, and then it will compute the overall rank of the website. This rank will determine whether the user's website contains any traces of terrorism. Using web mining and data mining, our system will detect patterns, keywords, and relevant information in unstructured text on a webpage. To retrieve textual content, our system will employ a web mining algorithm from web pages to recognize those that are relevant to the case. Web mining and data mining are sometimes used in tandem to achieve the best results. The application was built using Python and machine learning algorithms. The prospect of discovering This kind of web page can be recognized via using machine learning algorithms to scrutinize its content in real-time as discussed in this article. The client is the one who notices the issue. In this approach, suicidal people can be recognized early enough to recognize unsafe websites that the user visits frequently. A machine learning model is nothing more than a piece of code that has been trained with data by an engineer or data scientist. So, if you feed the model garbage, you'll receive garbage back, i.e. the trained model will forecast false or incorrect.

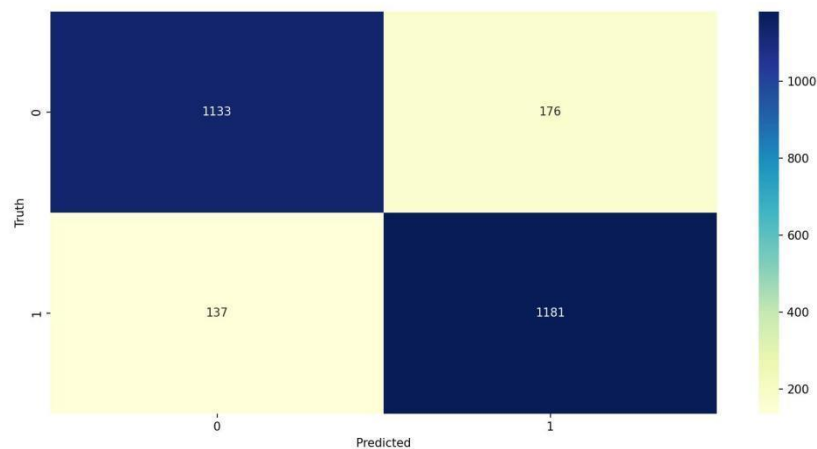


Fig1. Prediction graph

Beautiful Soup is a parsing package that performs a fantastic job at retrieving contents from URLs and allowing you to parse specific parts of them with ease. Web Scraping is also known as web data extraction, is a method of obtaining information from websites. The majority of this information is in the form of unstructured HTML that is converted to structured data in a spreadsheet or database before being used in various applications. It extracts data in a more comprehensible and hierarchical manner by generating a parse tree from the page source code. It's a complete web- scraping or crawling framework

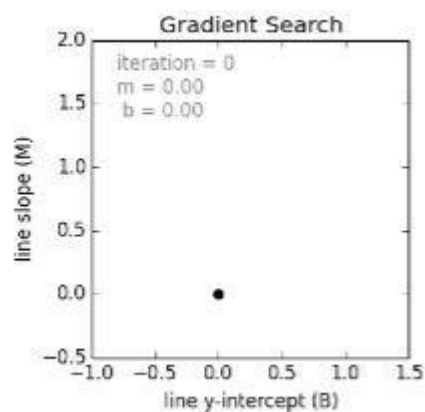


Fig2. Points and Line

Tkinter is a Python package that is frequently used to create graphical user interfaces. Tkinter makes creating a GUI a breeze, and the process is even faster. Tkinter provides a number of widgets that can be used to create a graphical user interface. Data collection enables us to keep a log of past details in order to use data analysis to uncover sequences. You can produce statistical models using machine learning algorithms that look for patterns and estimate future changes depending on those patterns. Good data collection processes are critical for generating high-performing models since predictive models are only as good as the data they're built on. The data must be devoid of errors (garbage in, garbage out) and contain information that is relevant to the work at hand. A debt default model, for example, would not profit from tiger population sizes but would benefit over time from gas costs. Much different statistical analysis and data

visualization approaches are used to investigate data in order to determine which data cleaning activities should be performed. We created a pre-processing function that helped lowercase the text, remove punctuation, and lemmatize related terms down to a single base word. After the pre-processing, the texts are evaluated using a machine learning algorithm that has been pre-trained to categorise themes possibly lethal or safe. The portion of the maximum of terms calculated by the algorithm is then computed. A pre-prepared training set was used for training and precision testing, which was gathered and classified based on certain texts into training and test data. The training dataset, that includes roughly 700.000 texts, and was used to train the models stored in nearly 2.000 texts, was used to evaluate the accuracy of the algorithms

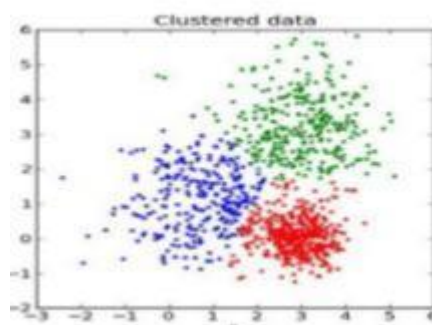


Fig3. Clustered data

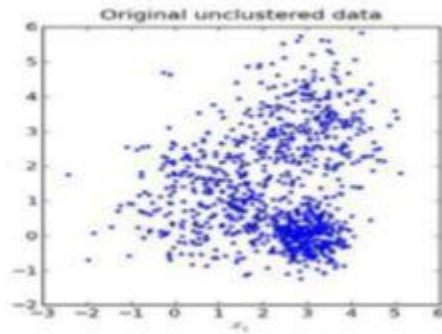


Fig4 .Original Unclustered data

To further comprehend our data, we employed a Count Vectorizer to examine the most frequently used words in each subedit (The words used were similar with some subtle differences). The changed text is then applied to various algorithms for training. Following that, a group on the web was constructed to test the algorithms under realistic conditions. Algorithms that are used are In its simplest form, logistic regression is a statistical method that applies a logistic function methodology is established to a binary dependent variable. There are numerous intricate augmentations In regression analysis, logistic regression (or logistic regression) is used to quantify a logistic model (a sort of binary regression). In statistical software, it is used to analyze possible outcomes and comprehend the correlation between the dependent and independent variables by using a logistic regression equation. This form of assessment can guide you in foretelling the likelihood of activity occurring or an intention being made. BeautifulSoup is a Python library for data extraction that retrieves data from HTML and XML files. It produces a parse tree from the page code base, which can be used to extract information in a more consolidated and readable manner. It is a complete framework for web-scraping or crawling. BeautifulSoup is a parsing library that also does a job of fetching contents from URLs and allowing you to easily parse specific parts of them. It only retrieves the contents of the URL you provide and then exits. Tkinter is a Python library that is commonly used to create graphical user interfaces (GUIs). It is very simple to create a GUI with Tkinter, and the process is even faster. Tkinter includes a number of widgets that can be used when creating a graphical user interface. These include buttons, radio buttons, checkboxes, etc The constant data process is performed in machine

learning, and Python's libraries allow you to access, handle, and modify information. These are the most extensively used libraries for ML and AI: Scikit-learn is used to handle simple machine learning algorithms such as clustering, linear and logistic regressions, regression, classification, and others.

A. Intuitions and design principles

A number of intuitions, which we list below, motivate the model we propose in this paper:

1) Malicious web pages often have small snippets of malicious content (such as malicious Javascript) embedded in some variable amount of benign content

(e.g. the original benign content in a hacked webpage). Identifying that a given document is malicious thus requires that a model examine the document at multiple

spatial scales. That is because the range of sizes of malicious Javascript snippets is small, but the variance in length of HTML documents is quite large, meaning

that proportion of the document length that represents malicious content is variable among examples.

2) Explicit parsing of HTML documents, which in reality are collections of HTML, Javascript, CSS, and raw data, is undesirable, because it significantly complicates implementation, can require high computational overhead, and opens up an attack surface within the detector itself which could potentially be exploited by attackers.

3) Emulation, static analysis, or symbolic execution of Javascript within HTML documents is undesirable because of the computational overhead it imposes and

because of the attack surface it opens up within the detector.

Following from these intuitions, we made the following high level design decisions in creating our proposed approach:

1) Instead of performing detailed parsing, static analysis, symbolic execution, or emulation of content within HTML documents, we compute a simple bag of-

words-style tokenization of documents that makes minimal assumptions about their constituent formats.

2) Instead of simply using a flat, bag-of-tokens representation aggregated over the entire document, we use a representation that captures locality at multiple spatial scales representing different levels of localization and aggregation, allowing our model to find needle-in-the-haystack malicious content within otherwise benign malicious documents.

Our approach involves a feature extractor, which parses out a sequence of tokens from HTML documents, and a neural network model, which makes classification decisions by using a shared-weight examination of the features at hierarchical levels of aggregation. The neural network includes two logical components:

- The first component is an *inspector*, which applies weights at hierarchical spatial scales and aggregates information about the document into a 1024-length vector.
- The second component is a *master network* which uses the output of the inspector network to make a final classification decision. The inspector and master components of the network are optimized jointly through back-propagation. Below we walk through how each of these logical components contribute to our overall system.

Our system's workflow begins by extracting a sequence of character-string tokens from HTML files. First we tokenize the target document with the following regular expression: `([^\x00-\x7F]+\w+)`, which splits the document along non-alphanumeric word boundaries. Then we divide the token stream up into 16 sequential chunks of equal length - where length is defined as number of tokens - including fewer tokens in the last chunk if the number of tokens in the documents is not divisible by 16. A reference implementation of the tokenization and chunking function is given as Python code in the Appendix of this paper under the heading *TokenizeChunk*.

Next we use a modified version of the hashing trick, with 1024 bins, to create a bag-of-words style representation for each chunk. We use a technique introduced in [10] to modify bin placement so that it's a function both of feature hash and token length (a Python reference implementation is given as *TokenLengthHash* in the Appendix). The result of this workflow, in which we tokenize, and then divide into 16 equal length token chunks, and then feature hash each token chunk into 1024 bins, is a 16x1024 tensor representing a sequence of bags-of-tokens extracted from an HTML document, where each element in the sequence represents an aggregation over a contiguous 1/16 of the input document.

Once we have a feature representation for an HTML document, we input that representation into our neural network, which we depict in Figure 1. As shown in the Figure, the first step in our computational flow is to create a hierarchical representation of our sequence of bags of tokens, in which we collapse our initial 16 bags of tokens into 8 bags of tokens, collapse these 8 bags of tokens into 4, those 4 into 2, and those 2 into 1, such that we obtain multiple bag-of-tokens representations capturing token occurrences at multiple spatial scales. This collapsing process works by averaging windows of length 2 and step size 2 over the original 16 bags of tokens, and then doing this recursively until we get to a single bag of tokens. An

important distinction to note here is that, by averaging, rather than summing, the token counts, we keep the norm of each representational level the same within a given document (though it can vary between documents of different overall length). Once the inspector has created this hierarchical representation, it proceeds to visit each node in the tree of aggregations and compute an output vector. the inspector is a feed forward neural network with two fully connected layers which each have 1024 ReLU units. We use layer normalization [11] to guard against vanishing gradients and dropout [12] to regularize the inspector. We use a dropout rate of 0.2. To compute a 1024-dimensional vector output from the inspector after it has visited each node, we take the maximum activation from each of its 1024 output neurons across the 31 outputs produced from the 31 aggregated chunks. This causes the final vector representation of the document to be the maximum output of each

neuron in the final layer of the inspector, given all of its activations over all nodes in the hierarchy. Intuitively, this should promote the output vector capturing patterns that most closely match known template features useful in predicting malicious content, regardless of where they appear in the document, or how long the overall document is.

Once the inspector has computed its 1024- dimensional output vector over the target document, this vector is input into the master component of our model. As shown the master is implemented as a feed forward neural network with two logical fully-connected blocks, where each fully connected layer is preceded by layer normalization and dropout. As in the case of the inspector, here we use a dropout rate of 0.2. The final layer of the model consists of 26 sigmoid units corresponding to 26 detection decisions we make about documents. One of these sigmoids is devoted to determining if the target document is malicious or benign. The other 25 sigmoids detect a variety of informative tags, such as whether the document is a phishing document or an instance of an exploit kit. To train the model, we use binary cross-entropy loss on each of these sigmoid outputs and then average the resulting gradients to compute parameter updates. In this paper, we emphasize evaluating the accuracy of the good vs. bad sigmoid output, but touch on our performance on these other outputs below as well.

We tested our approach in two ways. First, we compared it to a number of bag-of-words style baselines which represent a standard document classification approaches. Second, we modified the architecture in a variety of ways to test whether our model design choices were contributing to improved accuracy. We did not directly compare our approach to approaches that involve complex parsing of or emulation of web content, because the performance overhead of these approaches placee them out of scope of our research goals, which were to create a fast web content detection model that can operate on web content observed on firewalls and endpoints. Below we describe our experimental dataset, our baselines, and our model modifications and then introduce and discuss our results.

We tested our approach on data collected from the threat intelligence site VirusTotal. VirusTotal receives tens of thousands of new HTML files per day, scanning them with 60 web threat scanners from dozens of security vendors.

Sophos subscribes

to VirusTotal's paid threat intelligence service, and as part of this subscription we receive every HTML file submitted to VirusTotal with its corresponding scanner results. The experimental dataset used in this paper was collected from the VirusTotal feed in the first 10 months of 2017, as shown in Figure 2. Files were uniquely identified based on SHA256, and our training/testing splits were computed on the basis of the time that the file was first reported on VirusTotal. This process

ensures that a) our training and test sets are distinct (as a later submission of an identical HTML file would be interpreted as a resubmission of that file, and we would ignore it in favor of the earlier version) and b) that our training and testing

procedure at least approximates a real-world deployment scenario, as discussed in more detail below. Our labeling strategy is to use the scanner ensemble's response to a given file as the input to a labeling rule, which derives binary good versus bad labels from these data. Specifically, our labeling rule defines benign HTML files

as files receiving 0 detection's from security vendors, and malicious files as files receiving 3 or more detection's. Additionally, we discard files that received 1 or 2 detection's and did not use them in our research, because we consider this small number of detection's to mean that the security industry is still uncertain as to whether they are malicious or benign. As can be seen in Figure 2, these indeterminate files represent a very small proportion of the overall files from this time period. This labeling approach carries the risk that our approach will simply memorize the knowledge of security vendor products, as opposed to learning truly novel detection capabilities that detect malware the vendor community would have missed. In this work, we test this using a historical simulation procedure, defined as follows:

- 1) We train our models on web content files first seen in the VirusTotal feed before some time t .

2) We evaluate our models on web content files first seen in the two months after time t , using the latest vendor labels. This evaluation procedure mitigates the problem of assessing our ability to detect malicious web content the vendor community would have missed. This is because we test on files we haven't seen in training, but which the security vendor community has time to catch up to through detection rule and blacklist updates, thereby simulating the problem of detecting 0-day malicious content.

In other words, insofar as a machine learning model is able to correctly predict the label of future web content the vendor community has had time to blacklist or write detection rules for, we believe there is at least circumstantial evidence that

our approach also has the ability to detect malicious web content the vendor community can be expected to miss entirely. To provide more direct evidence of the efficacy of our proposed approach, we also hand-inspected samples that our labeling strategy marked as benign but for which our proposed model assigned a high probability of maliciousness, finding that a majority of these supposed false positives were either clearly malicious or junk content. We discuss these results below.

Aside from evaluating our proposed model, we conducted five additional experiments, three of which test the efficacy of our approach against alternative models, and two of which explore the inner workings of our model.

To test the effi-

cacy of our model architecture we conducted the following three experiments. For all neural network models, models were trained using the Adam optimization scheme, balanced batches of size 64, and early stopping based on validation set performance. We enumerate our test approaches below: *LR-BoT*: Elastic Net regularized logistic regression on bag of token features. Here we used the tokens we extract for

our full model, but feature hash them into a 16284-length vector. We chose to use a 16284 input in this and other baseline experiments since it is the same dimensionality as the 16x1024 representation we use in our proposed model, and thus approximates an apples-to-apples comparison. The LR-BoT test model provides a straightforward bag-of-tokens linear document classification

approach as a point of comparison. We determined the λ_1 and λ_2 parameters for the L1 and L2 penalties via grid search.

FF-BoT: A feed-forward architecture using the 16284-length feature hashed bag of tokens feature representation used above. This test model provides a straightforward deep learning bag-of-words baseline as a point of comparison with our proposed approach.

XGBoost-BoT: A gradient boosted decision tree (XG-Boost) model using the same feature input as FF-BoT. Separate from our baseline experiments, we performed several architecture modifications to determine the contribution of the design decisions we made in creating our proposed model, which we list here:

FlatSequential: A variant of our proposed model with the average pooling step removed, such that the inspector only sees the leaf nodes of the tree. Put another way, out of the 31 aggregated representations input to the champion model, this model only sees the 16 sequential chunks, and none of the larger aggregated windows. This model tests the gain in performance provided by examining HTML documents at hierarchical spatial scales.

FlattenedFF: A simple feed forward neural network that uses the same feature representation as our selected model. However, rather than applying the shared-weight inspector at each step, we simply rasterize the 16x1024 sequential bag of tokens vectors into a single, 16284-length vector and input that into a feed forward neural network. This experiment evaluates the performance gain of using a shared-weight inspector at all, relative to a dense first layer.

CHAPTER 4

4.1 RESULT

In this concept design study, we suggested a technique for creating a classification model for precedent suicidal ideation among adolescents using natural language processing and machine learning of clinical narratives from health record data prior to admission. This is the first time that unstructured data has been used to up skill a machine-learning classification system in a teenage inpatient population using NLP as far as we know. This first success signal based on a limited sample size needs to be confirmed in bigger datasets, The method demonstrates how EHR notes for adolescent suicide attempts can be enhanced with clinically relevant information to identify children with a history of suicide risk, This can aid in patient's health organizing during a highly vulnerable period for this high-risk demographic.



Fig5. Select the Url

Detect

Enter a webpage

Select your text file containing urls

https://suicidepreventionlifeline.org/ = 0

Keywords matched:

[]

Fig6.Detects without Suicidal content the url

Detect

Enter a webpage

https://suicideprevention.nv.gov/Youth/Myths/

Select your text file containing urls

Fig7.Select the Url

Detect

Enter a webpage

Select your text file containing urls

https://suicideprevention.nv.gov/Youth/Myths/ = 1

Keywords matched:

[["killing", 1]]

Fig8 .Detects with Suicidal content the url

4.2 PERFORMANCE ANALYSIS

ALGORITHM	FUNCTION	LIMITATION	PERFORMANCE ANALYSIS
Rotation Forest algorithm and	Maximum Probability voting classification decision method	This approach works best with users who mention their age.	Accuracy is high
Support Vector Machines and Logistic Regression	Cross-validation method	The efficacy of Twitter for actual suicide ideation is unidentified, and the study results do not directly describe intervention targets.	Accuracy score of 76%
Support Vector Machines (SVMs) with Term Frequency weighted by Inverse Document Frequency	Naive Bayes algorithm	There is no connection between the timing of suicidal tweets and real suicide.	Achieved a performance of 0.657

Transfer learning	Discrete distance decay function	DNN-based method for the prediction of crime occurrences cannot be applied when sufficient data is unavailable.	DNN model is more accurate
-------------------	----------------------------------	---	----------------------------

CHAPTER 5

5.1 FUTURE SCOPE

To curb the menace of terrorism and to destroy the online presence of dangerous terrorist organizations like ISIS and other radicalization websites. We need a proper system to detect and terminate websites which are spreading harmful content used to radicalizing youth and helpless people. We analysed the usage of Online Social Networks (OSNs) in the event of a terrorist attack. We used different metrics like number of tweets, whether users in developing countries tended to tweet, re-tweet or reply, demographics, geo location and we defined new metrics (reach and impression of the tweet) and presented their models. While the developing countries are faced by many limitations in using OSNs such as unreliable power and poor Internet connection, still the study finding challenges the traditional media of reporting during disasters like terrorist's attacks. We recommend centers globally to make full use of the OSNs for crisis communication in order to save more lives during such

5.2 CONCLUSION

This paper created a cyber-bullying detection model using TF-IDF and Word2Vec feature extraction. Several machine-learning-based text classification approaches were also investigated. The tests were carried out with the help of a global Twitter dataset. According to the analytical outcomes, LR has the highest accuracy rate and F1 score in our dataset. With 90.57 percent classification accuracy and 0.9280 F1 scores, respectively. Meanwhile, the performance of the LR, SGD and LGBM classifiers differs slightly, with SGD achieving 90.6 percent accuracy but a lower F1 score than LR. The LGBM classifier, on the other hand, had an accuracy of 90.55 percent and an F1 score of 0.9271. This indicates that LR outperforms other classifiers. Furthermore, it was discovered during the trials that LR performs better as data size grows and when compared to the other classifiers used in this study, has the shortest prediction time. As a result, SGD performs nearly

as well as LR, although the error is not as small as it is in LR. Feature extraction is an important part of machine learning for improving detection precision we did not examine many extracting features methods in this study. one more area for improvement to achieve a more accurate rate of both the LR and SGD classifiers is to integrate and compare different feature extractions. Another constraint on which we are working is the creation of a real-time cyber bully prediction tool that will aid in the detection and prevention of cyber bullies in real-time. Another line of investigation is cyber bully detection in many languages, with a focus on Arabic.

REFERENCES

- [1] Lyovkin, Maxim, Aleksey A. Frolov, and Egor Perminov. "Detection of Dangerous Web Pages Based on the Analysis of Suicidal Content Using Machine Learning Algorithms."2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus). IEEE, 2021.
- [2] Goradia, Rinkle, et al. "Web Mining to Detect Online Spread of Terrorism."International Journal of Research & Technology (IJERT) 9.7 (2020): 645- 648.
- [3] Li, Xue, Dongmei Zhang, and Bin Wu. "Detection method of phishing email based on persuasion principle."2020,IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC).Vol. 1. IEEE, 2020.
- [4]Mbarek, Atika, et al. "Suicidal Profiles Detection in Twitter."WEBSITE. 2019.
- [5] Kawaguchi, Yuki, and Seiichi Ozawa. "Exploring and identifying malicious sites in dark web using machine learning."International Conference on Neural Information Processing. Springer, Cham, 2019.
- [6] NARYNOV, S., MUKHTARKHANULY, D., KERIMOV, I. and OMAROV, B., 2019."Comparative analysis of supervised and unsupervised learning algorithms for online user content suicidal ideation detection.", Journal of Theoretical and Applied Information Technology, 97(22), pp.3304-3317.
- [7] Saxe J, Harang R, Wild C, Sanders H. "A deep learning approach to fast, format-agnostic detection of malicious web content". In2018 IEEE Security and Privacy Workshops (SPW) 2018 May 24 (pp. 8-14). IEEE.
- [8] Madhusudhan, R. "Cross Channel Scripting (XCS) Attacks in Web Applications: Detection and Mitigation Approaches."2018 2nd Cyber Security in Networking Conference (CSNet). IEEE, 2018.
- [9] Buber, E., Demir, Ö., & Sahingoz, O. K. (2017, September). Feature selections for the machine learning- based detection of phishing websites. In 2017, international artificial intelligence and data processing symposium (IDAP) (pp. 1-5). IEEE.

- [10] Gerber, Matthew S. "Predicting crime using Twitter and kernel density estimation." *Decision Support Systems* 61 (2014): 115-125.
- [11] Shingleton, Jarrod S. Crime trend prediction using regression models for salinas, California. NAVAL POSTGRADUATE SCHOOL MONTEREY CA, 2012.
- [12] Hattori, G., Matsumoto, K., Ono, C., & Takishima, Y. (2010, October). Identification of malicious web pages for crawling based on network-related attributes of the webserver. In 2010 4th International Universal Communication Symposium (pp. 355-361). IEEE.
- [13] Choi, Y., Kim, T., Choi, S. and Lee, C., 2009, December. Automatic detection for javascript obfuscation attacks in web pages through string pattern analysis. In International Conference on Future Generation Information Technology (pp. 160-172). Springer, Berlin, Heidelberg.
- [14] Seifert, Christian, Ian Welch, Peter Komisarczuk, Chiraag Uday Aval, and Barbara Endicott-Popovsky. "Identification of malicious web pages through analysis of underlying DNS and web server relationships.", In 2008 33rd IEEE Conference on Local Computer Networks (LCN), pp. 935-941. IEEE, 2008.
- [15] Felix, A. Yovan, and T. Sasipraba. "Decision support system for flood risk assessment and public sector performance management of emergency scenarios." *International Journal of Public Sector Performance Management* 8, no. 3 (2021): 219-229

6.2.APPENDIX

6.2.1 SAMPLE CODE

```
import subprocess
import sys

subprocess.check_call([sys.executable, '-m', 'pip', 'install', 'requests'])
subprocess.check_call([sys.executable, '-m', 'pip', 'install', 'bs4'])
subprocess.check_call([sys.executable, '-m', 'pip', 'install', 'ttkthemes'])
subprocess.check_call([sys.executable, '-m', 'pip', 'install', 'lxml'])
subprocess.check_call([sys.executable, '-m', 'pip', 'install', 'tkinter'])
subprocess.check_call([sys.executable, '-m', 'pip', 'install', 'PIL'])
import model
import preprocess as pp

shouldPreprocess = False

if __name__ == '__main__':
    if shouldPreprocess:
        data = pp.getDataset('RedditDataSentimental.xlsx')
        data = pp.cleanDataset(data, True, True)

        # Get preprocessed dataset
        data = pp.getDataset('dataset.csv')
        data = model.minSubSample(data, False)

        # Indication for Logistic Regression model
        if False:
            model.validateModel(data) # not executed except for testing due to time
            required to evaluate

        # Parameter tweaking to trade accuracy for greater positive recall /
        sensitivity
        if False:
            model.gridSearchParameters(data) # not executed except for testing due to
            time required to evaluate

        # Final training
        model.trainFinalModel(data)
import math

from sklearn.metrics import accuracy_score
from sklearn.svm import LinearSVC

import pipeline as pl
```

```

import preprocess as pp
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
from pprint import pprint
from sklearn.utils import shuffle
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn import metrics

def minSubSample(data, shouldShuffle=False):
    """
    minSubSample - prepares DataFrame by taking ?shuffled subsample from
    data, based on minimum size of groups, to balance classes

    :param data: pandas DataFrame
    :param shouldLemmatise: boolean, defaults False - determines whether
    groups are shuffled
    :return: returns DataFrame
    """
    # Get smallest sample group index
    minIndex = np.argmin(data['Suicidal'].value_counts())

    # Separate into distinct groups
    suicidalGroup = data[data['Suicidal'] == 1]
    notSuicidal = data[data['Suicidal'] == 0]
    suicidalGroup = suicidalGroup.reset_index(drop=True)
    notSuicidal = notSuicidal.reset_index(drop=True)

    # Shuffle if required
    if shouldShuffle:
        suicidalGroup = shuffle(suicidalGroup)
        notSuicidal = shuffle(notSuicidal)

    # Sample groups by smallest group size
    numSamples = len([notSuicidal, suicidalGroup][minIndex])
    notSuicidal = notSuicidal[:numSamples]
    suicidalGroup = suicidalGroup[:numSamples]

    # Merge groups of data and shuffle
    data = shuffle(pd.concat([suicidalGroup,
    notSuicidal])).reset_index(drop=True)

    return data

def validateModel(data):

```

```

"""
    validateModel - tests validity and prints accuracy score of Logistic
    Regression model

    :param data: pandas DataFrame
    :return: None
    """

    # Tfidf transformation
    tfidfVect = TfidfVectorizer()
    trainContentTfidf = tfidfVect.fit_transform(data['Sentence'])

    # Fit model
    skf = StratifiedKFold(n_splits=5, shuffle=True)
    skf.get_n_splits(trainContentTfidf, data['Suicidal'])

    LR = LogisticRegression()
    scores = []
    for train_index, test_index in skf.split(trainContentTfidf, data['Suicidal']):
        x_train, x_test = trainContentTfidf[train_index], trainContentTfidf[test_index]
        y_train, y_test = data.iloc[train_index, 1], data.iloc[test_index, 1]

        LR.fit(x_train, y_train)

        y_pred = LR.predict(x_test)
        scores.append(round(metrics.accuracy_score(y_test, y_pred) * 100, 2))
        # print(metrics.classification_report(y_test, y_pred, target_names=['Not
        Suicidal', 'Suicidal']))

    # Print scores
    scores = [{'Split': i, 'Accuracy': x} for i, x in enumerate(scores)]
    print('_____LR indication_____\n')
    pprint(scores)
    print('_____')

def evaluateModel(pipeline, train_data, train_labels):
    """
        evaluateModel - evaluate pipeline model from pl

        :param pipeline: pipeline referring to MODEL_PIPELINES index
        :param train_data: x_train data
        :param train_labels: y_train data
        :return: return GridSearchCV dataframe
        """

    # Get the current pipeline and its parameters
    curPipeline = pl.MODEL_PIPELINES[pipeline]['Pipeline']
    curPipelineParams = pl.MODEL_PIPELINES[pipeline]['Parameters']

    # Grid search cv to evaluate the models for scorers
    gs_eval = GridSearchCV(
        curPipeline,

```

```

    curPipelineParams,
    scoring=pl.MODEL_SCORERS,
    refit=pl.REFIT_SCORE,
    error_score='raise',
    cv=StratifiedKFold(n_splits=10, random_state=None),
    n_jobs=-1
)
gs_eval = gs_eval.fit(train_data, train_labels)

print(f'Best params for {pl.REFIT_SCORE}')
print(gs_eval.best_params_)

return gs_eval

def evaluateModels(train_data, train_labels):
    """
    evaluateModels - evaluate all models and parameters within
    pl.MODEL_PIPELINES

    :param train_data: x_train data
    :param train_labels: y_train data
    :return: array (names of pipelines), pd DataFrame (cv_results_), array
    (best_params_)
    """
    names = []; results = []; params = []
    for curPipeline in pl.MODEL_PIPELINES:
        gs = evaluateModel(curPipeline, train_data, train_labels)
        names.append(curPipeline)
        results.append(pd.DataFrame(gs.cv_results_))
        params.append(gs.best_params_)

    return names, results, params

def gridSearchParameters(data):
    """
    gridSearchParameters - hyper parameter tweaking as per pl model pipelines
    to trade accuracy for higher sensitivity

    :param data: pandas DataFrame
    :return: None
    """
    names, results, params = evaluateModels(data.iloc[:, 0], data.iloc[:, 1])

    for i, x in enumerate(names):
        print('Test -->' + x)
        print('Results -->')
        print(results[i].head())

def trainFinalModel(data):
    """

```

```

trainFinalModel - final training model with best parameters

:param data: pandas DataFrame
:return: None
"""

# Transform
tfidfVect = TfidfVectorizer(ngram_range=(1, 3), norm='l1')
trainContentTfidf = tfidfVect.fit_transform(data['Sentence'])

x_train, x_test, y_train, y_test = train_test_split(trainContentTfidf,
data.iloc[:,1], test_size=0.2)

# Fit model
LR = LogisticRegression(C=4, max_iter=1000, penalty='l2')
LR.fit(x_train, y_train)

# Predict
y_pred = LR.predict(x_test)
print(f' Accuracy of LR : {round(metrics.accuracy_score(y_test, y_pred) *
100, 2)}%')
print(metrics.classification_report(y_test, y_pred))
LogisticRegression_score = metrics.accuracy_score(y_test, y_pred)

# Visualise model prediction vs truth
sns.heatmap(metrics.confusion_matrix(y_test, y_pred), annot=True,
fmt='d', cmap='YlGnBu')
plt.title('Confusion matrix of logistic regression', y=1.1)
plt.xlabel('Predicted')
plt.ylabel('Truth')
plt.show()

# Fit model
classifier = LinearSVC(C=4, max_iter=1000, penalty='l2')
classifier.fit(x_train, y_train)

# Predict
y_prediction = classifier.predict(x_test)
print(f' Accuracy Score of SVC: {round(metrics.accuracy_score(y_test,
y_prediction) * 100, 2)}%')
print(metrics.classification_report(y_test, y_prediction))
ACC1=metrics.accuracy_score(y_test, y_prediction)
# Visualise model prediction vs truth
sns.heatmap(metrics.confusion_matrix(y_test, y_prediction), annot=True,
fmt='d', cmap='YlGnBu')
plt.xlabel('Predicted')
plt.ylabel('Truth')
plt.title('Confusion matrix of SVC', y=1.1)
plt.show()

labels = ["SVC", "LogisticRegression"]

```

```

usages = [ACC1, LogisticRegression_score]
y_positions = range(len(labels))
plt.bar(y_positions, usages)
plt.xticks(y_positions, labels)
plt.ylabel("Accuracy")
plt.title("best model selection")
plt.show()
from texthero import preprocessing as pp
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import make_scorer, recall_score, accuracy_score,
precision_score

# File directory for training input
INPUT_DIRECTORY = './input/'

# Preprocessing pipeline for cleaning text
PREPROCESSING_PIPELINE = [
    pp.remove_html_tags,
    pp.remove_urls,
    pp.lowercase,
    pp.remove_digits,
    pp.remove_punctuation,
    pp.remove_diacritics,
    pp.remove_stopwords,
    pp.remove_whitespace,
    pp.remove_brackets
]

# Model scorers
REFIT_SCORE = 'recall_score'
MODEL_SCORERS = {
    'precision_score': make_scorer(precision_score),
    'recall_score': make_scorer(recall_score),
    'accuracy_score': make_scorer(accuracy_score)
}

# GridSearchCV pipelines
MODEL_PIPELINES = {
    'fe=tfidf+bigram+default_c=LR' : {
        'Pipeline' : Pipeline([
            ('tfidf', TfidfVectorizer()),
            ('clf', LogisticRegression())
        ]),
        'Parameters' : {
            'tfidf__norm' : ['l1', 'l2'],
            'tfidf__ngram_range': [(1, 2)],
            'clf__penalty': ['l2'],

```

```

        'clf__max_iter': [1000],
        'clf__C' : [0.25, 0.5, 1, 2, 4]
    }
},
'fe=tfidf+trigram+default_c=LR' : {
    'Pipeline' : Pipeline([
        ('tfidf', TfidfVectorizer()),
        ('clf', LogisticRegression())
    ]),
    'Parameters' : {
        'tfidf__norm' : ['l1', 'l2'],
        'tfidf__ngram_range': [(1, 3)],
        'clf__penalty': ['l2'],
        'clf__max_iter': [1000],
        'clf__C' : [0.25, 0.5, 1, 2, 4]
    }
}
}
import os
import emoji
import nltk
import pipeline as pl
import texthero as hero
import pandas as pd
import regex as re
from nltk.stem import WordNetLemmatizer

# Utilities
def getDataset(filename):
    """
    getDataset - gets and reads the specified file from the input directory,
    drops null rows and columns excluded from our analysis.

    :param filename: in format of {filename.extension}
    :return: returns pandas DataFrame
    """
    # Read file
    filename = pl.INPUT_DIRECTORY + filename
    split = os.path.splitext(filename)
    data = None
    if 'csv' in split[1]:
        data = pd.read_csv(filename)
    elif 'xlsx' in split[1]:
        data = pd.read_excel(filename)

    # Select only Suicidal and Sentence columns
    data = data[['Sentence', 'Suicidal']]

    # Remove null rows
    data = data.dropna()

```

```

return data

def removeHandles(text):
    """
    removeHandles - removes @alphanumeric handles from text

    :param text: string
    :return: returns cleansed string
    """
    text = ' '.join(re.sub('@\w+', '', text).split())
    return text

def removeHashtag(text):
    """
    removeHashtag - removes #hashtag from text

    :param text: string
    :return: returns cleansed string
    """
    text = ' '.join(re.sub('\#[\w_]+', '', text).split())
    return text

def removeEmoji(text):
    """
    removeEmoji - removes all instances of emoji in the string passed.

    :param text: string
    :return: returns cleansed string
    """
    text = emoji.get_emoji_regexp().sub(u'', text)
    return text

def cleanSentences(text, lemmatise=True):
    """
    cleanSentences - pre-processes text for analysis, optional lemmatisation
    (defaults to True)

    :param text: array of strings
    :param lemmatise: boolean, defaults True - determines whether function
    performs lemmatisation
    :return: returns array of cleaned strings
    """
    # Remove emoji
    text = text.apply(lambda x: ' '.join([removeEmoji(word) for word in x.split()]))

    # Remove hashtag
    text = text.apply(lambda x: ' '.join([removeHashtag(word) for word in
    x.split()]))

```

```

# Remove handles
text = text.apply(lambda x: ' '.join([removeHandles(word) for word in
x.split()])))

# Clean text as per the pipeline
text = hero.clean(text, pipeline=pl.PREPROCESSING_PIPELINE)

# Lemmatisation
if lemmatise:
    nltk.download('wordnet')
    text = text.apply(lambda x: ' '.join([WordNetLemmatizer().lemmatize(word)
for word in x.split()])))

# Remove words with < 2 len and/or aren't alphabetical
text = text.apply(lambda x: ' '.join([word for word in x.split() if len(word) > 2
and word.isalpha()])))

return text

def cleanDataset(data, shouldLemmatise=True, shouldSave=False):
    """
    cleanDataset - prepares DataFrame for analysis

    :param data: pandas DataFrame
    :param shouldLemmatise: boolean, defaults to True - passed to
    cleanSentences if lemmatisation required
    :param shouldSave: boolean, defaults to False - determines whether we
    save the cleaned DF
                                as a csv to ./inputs/dataset.csv
    :return: returns cleansed DataFrame
    """

    # Replace yes/no string datapoint as binary representation
    data.loc[data['Suicidal'] == 'Yes', 'Suicidal'] = 1
    data.loc[data['Suicidal'] == 'No', 'Suicidal'] = 0

    # Preprocess string data
    data['Sentence'] = cleanSentences(data['Sentence'], shouldLemmatise)

    # Save input if required
    if shouldSave:
        data.to_csv(pl.INPUT_DIRECTORY + 'dataset.csv')

    return data
import json # to print list/dict in textbox
import tkinter as tk # root GUI module
import tkinter.scrolledtext as scrolledtext # module for scrollable text widget
import tkinter.ttk as ttk # themed GUI module
from tkinter.filedialog import askopenfile # module to read file

import requests # module to get all contents of a website

```

```
from bs4 import BeautifulSoup # module to get only text from a website
from PIL import Image, ImageTk # module to open and load a image
from ttkthemes import ThemedStyle # module to use in-built GUI themes
```

```
# class to get all frames together
```

```
class MyApp(tk.Tk):
```

```
    def __init__(self, *args, **kwargs, ):
        tk.Tk.__init__(self, *args, **kwargs)
        container = tk.Frame(self)
        container.pack(side="top", fill="both", expand=True)
```

```
        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)
```

```
        self.frames = {}
```

```
        menu = tk.Menu(container)
```

```
        ex = tk.Menu(menu, tearoff=0)
        menu.add_cascade(menu=ex, label="Exit")
        ex.add_command(label="Exit",
                        command=self.destroy)
```

```
        tk.Tk.config(self, menu=menu)
```

```
        for F in (Startpage, PageOne, PageTwo):
            frame = F(container, self)
            self.frames[F] = frame
            frame.grid(row=0, column=0, sticky="nsew")
```

```
        self.show_frame(Startpage)
```

```
    def show_frame(self, cont):
        frame = self.frames[cont]
        frame.tkraise()
```

```
# Home page
```

```
class Startpage(ttk.Frame):
```

```
    def __init__(self, parent, controller):
        ttk.Frame.__init__(self, parent)
```

```
        label = ttk.Label(self, text="Detection of suicidal content",
font=("Simplifica", 22)) # page heading
        label.pack(pady=5, padx=5)
```

```
        ttk.Label(self, text="").pack()
```

```

        button1 = ttk.Button(self, text="Detect",
                               command=lambda: controller.show_frame(PageOne)) #
got to detect page
        button1.pack()

        ttk.Label(self, text="").pack()

        button2 = ttk.Button(self, text="About",
                               command=lambda: controller.show_frame(PageTwo)) #
got to about page
        button2.pack()

        ttk.Label(self, text="").pack()

        img = ImageTk.PhotoImage(Image.open(r'wallpaper.jpg').resize((1200,
700))) # set the home page image
        img.image = img
        ttk.Label(self, image=img).pack()

# ***** PAGES *****
# Detect page
class PageOne(ttk.Frame):

    def __init__(self, parent, controller):
        ttk.Frame.__init__(self, parent)
        label = ttk.Label(self, text="Detect", font=("Simplifica", 22)) # page
heading
        label.pack(pady=5, padx=5)

        ttk.Label(self, text="\n").pack()

        ttk.Label(self, text="Enter a webpage", font=(18)).pack()
        text = tk.Entry(self, font=(26), width=70, bg="lightgray") # textbox to
enter a website
        text.pack()

        ttk.Label(self, text="").pack()

        # load the file containing fixed keywords
        j = []
        f = open(r'keywords.txt')
        for line in f:
            j.append(line.strip())
        f.close()
        d = dict.fromkeys(j, 0)

        # code to scan the website given in textbox
        def scan():

```

```

count = 0
url = text.get()
text.delete(0, "end")
result = requests.get(url.strip())
soup = BeautifulSoup(result.content, 'lxml')
for i in soup.get_text().split():
    if (i.lower() in j):
        count += 1
    if i.lower() in d:
        d[i.lower()] += 1
l3.config(state=tk.NORMAL)
l3.delete('1.0', "end")
di = dict(sorted(d.items(), reverse=True, key=lambda item: item[1]))
lis = [(k, v) for k, v in di.items() if v >= 1]
l3.insert(tk.END, url.strip() + " = " + str(count) + "\n\nKeywords
matched: \n" + json.dumps(lis))
l3.config(state=tk.DISABLED)

b2 = ttk.Button(self, text="Scan", command=scan)
b2.pack()

ttk.Label(self, text="").pack()

# code to open and scan the list of websites given in a text file
def open_n_scan():
    files = askopenfile(mode='r', filetypes=[("Text File", "*.txt")])
    l3.config(state=tk.NORMAL)
    l3.delete('1.0', "end")
    for url in files:
        count = 0
        result = requests.get(url.strip())
        soup = BeautifulSoup(result.content, 'lxml')
        for i in soup.get_text().split():
            if (i.lower() in j):
                count += 1
        l3.insert(tk.END, url.strip() + " = " + str(count) + "\n")
    l3.config(state=tk.DISABLED)

ttk.Label(self, text="Select your text file containing urls", font=(18)).pack()

b1 = ttk.Button(self, text="Open and Scan", command=open_n_scan)
b1.pack()

ttk.Label(self, text="").pack()

l3 = scrolledtext.ScrolledText(self, font=(18), height=10, width=70,
bg="lightgray",
                                state=tk.DISABLED) # multiline textbox
l3.pack()

```

```

        ttk.Label(self, text="").pack()

        button1 = ttk.Button(self, text="Back to Home",
                               command=lambda: controller.show_frame(Startpage)) # go
to home page
        button1.pack()

        ttk.Label(self, text="").pack()

        button2 = ttk.Button(self, text="About",
                               command=lambda: controller.show_frame(PageTwo)) #
got to about page
        button2.pack()

# About page
class PageTwo(ttk.Frame):

    def __init__(self, parent, controller):
        ttk.Frame.__init__(self, parent)
        label = ttk.Label(self, text="About", font=("Simplifica", 22)) # page
heading
        label.pack(pady=5, padx=5)

        ttk.Label(self, text="").pack()

        button1 = ttk.Button(self, text="Back to Home",
                               command=lambda: controller.show_frame(Startpage)) #
got to home page
        button1.pack()

        ttk.Label(self, text="").pack()

        button2 = ttk.Button(self, text="Detect",
                               command=lambda: controller.show_frame(PageOne)) #
got to detect page
        button2.pack()

        ttk.Label(self, text="").pack()

        tk.Message(self, relief="sunken", bd=4, font=(20), width=1100,
                    text="Suicide is a serious public health problem that can have
long-lasting effects on individuals, families, and communities. The good news
is that suicide is preventable. Preventing suicide requires strategies at all
levels of society. This includes prevention and protective strategies for
individuals, families, and communities. Everyone can help prevent suicide by
learning the warning signs, promoting prevention and resilience, and a
committing to social change.").pack()
        # Info on suicide
        ttk.Label(self, text="").pack()

```

```

tk.Message(self, relief="sunken", bd=4, font=(20), width=1100,
            text=" Contact the National Suicide Prevention Lifeline Call 1-800-273-TALK (1-800-273-8255) You'll be connected to a skilled, trained counselor in your area. For more information, visit the National Suicide Prevention Lifelineexternal icon").pack()
# Info on suicide content
ttk.Label(self, text="").pack()

tk.Message(self, relief="sunken", bd=4, font=(20), width=1100,
            text=" Coping and problem-solving skills, Cultural and religious beliefs that discourage suicide, Connections to friends, family, and community support, Supportive relationships with care providers, Availability of physical and mental health care, Limited access to lethal means among people at risk").pack()
# About
ttk.Label(self, text="").pack()

ttk.Label(self, text="", font=(20)).pack() # copyright

app = MyApp()

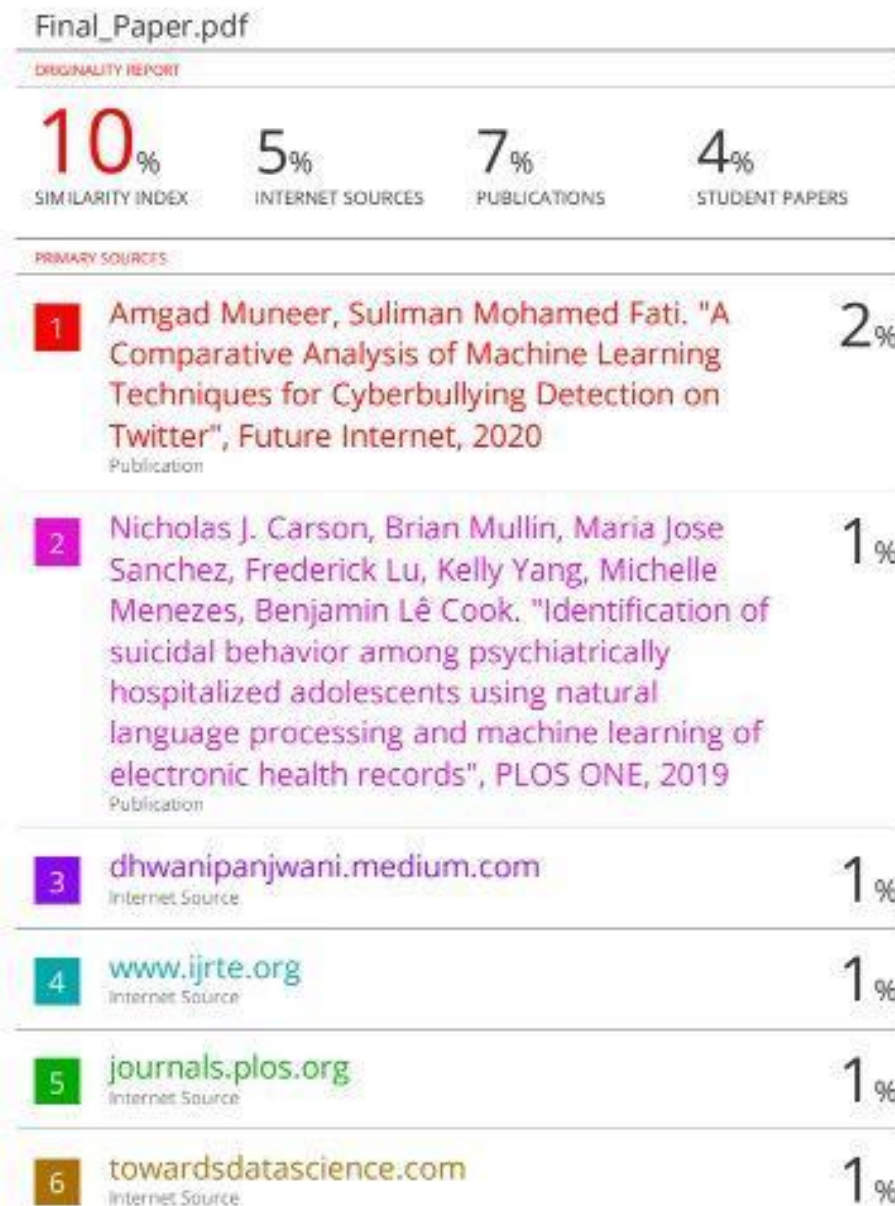
# set default app theme
style = ThemedStyle(app)
style.set_theme("plastik")

# set app icon
icon = ImageTk.PhotoImage(Image.open(r'icon.jpg'))
app.iconphoto(True, icon)

app.resizable(0, 0)
app.title("Detect web pages with suicidal content") # app title
app.state('zoomed') # maximized app by default
app.mainloop()

```

PUBLICATION REPORT :



7	Maxim Lyovkin, Aleksey A. Frolov, Egor Perminov. "Detection of Dangerous Web Pages Based on the Analysis of Suicidal Content Using Machine Learning Algorithms", 2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus), 2021 Publication	1 %
8	analyticsindiamag.com Internet Source	1 %
9	Hattori, Gen, Kazunori Matsumoto, Chihiro Ono, and Yasuhiro Takishima. "Identification of malicious web pages for crawling based on network-related attributes of web server", 2010 4th International Universal Communication Symposium, 2010. Publication	1 %
10	Submitted to Higher Education Commission Pakistan Student Paper	<1 %
11	Theses.ubn.ru.nl Internet Source	<1 %
12	hrcak.srce.hr Internet Source	<1 %
13	Hyeon-Woo Kang, Hang-Bong Kang. "Prediction of crime occurrence from multi-	<1 %

CERTIFICATE



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with Grade 'A' By NAAC | 12B Status by UGC | Approved By AICTE
www.sathyabama.ac.in

School of Computing
Department of Information Technology

Certificate of Presentation

This is to certify that

M. DHARSHINI NITHILA

of

Sathyabama Institute of Science and Technology

has presented a paper entitled

Predicting Harmful Web Pages Based on Suicide - related Textual Analysis using Machine Learning Algorithms

in the AICTE sponsored
International Conference on Artificial Intelligence and Machine Learning (IAIM-2022),
held from 27th January 2022 - 29th January 2022.


Dr. R. SUBHASHINI
HOD [Information Technology]


Dr. T. SASIKALA
Dean [School of Computing]


Dr. T. SASIPRABA
Vice Chancellor



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with Grade 'A' By NAAC | 12B Status by UGC | Approved By AICTE
www.sathyabama.ac.in

School of Computing
Department of Information Technology

Certificate of Presentation

This is to certify that

VINISHA R

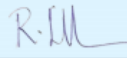
of

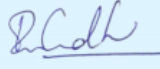
Sathyabama Institute of Science and Technology

has presented a paper entitled

Predicting Harmful Web Pages Based on Suicide - related Textual Analysis using Machine Learning Algorithms

in the AICTE sponsored
International Conference on Artificial Intelligence and Machine Learning (IAIM-2022),
held from 27th January 2022 - 29th January 2022.


Dr. R. SUBHASHINI
HOD [Information Technology]


Dr. T. SASIKALA
Dean [School of Computing]


Dr. T. SASIPRABA
Vice Chancellor