SUICIDE IDEATION DETECTION IN SOCIAL MEDIA FORUMS

MAJOR PROJECT REPORT submitted in partial fulfillment of the requirements

Submitted by

VISHAL DESU 178W1A1273

NIKHILESWAR KOMATI 178W1A1294

SPHOORTHI LINGAMANENI 178W1A1299

Under the Guidance of

Dr. Shaik Fathimabi, Ph.D

Sr. Assistant Professor

For the award of the degree

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

DEPARTMENT OF INFORMATION TECHNOLOGY

V R SIDDHARTHA ENGINEERING COLLEGE

(AUTONOMOUS - AFFILIATED TO JNTU-K, KAKINADA)

Approved by AICTE &Accreted by NBA

KANURU, VIJAYAWADA-7

ACADEMIC YEAR (2020-21)

V.R.SIDDHARTHA ENGINEERING COLLEGE

(Affiliated to JNTUK: Kakinada, Approved by AICTE, Autonomous)
(An ISO certified and NBA accredited institution)
Kanuru, Vijayawada – 520007

CERTIFICATE

This is to certify that this major project report titled **"SUICIDE IDEATION DETECTION IN SOCIAL MEDIA FORUMS"** is a bonafide record of work done by **VISHAL.D (178W1A1273), NIKHILESWAR.K (178W1A1294), and SPHOORTHI.L (178W1A1299)** under my guidance and supervision is submitted in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Information Technology, **V.R. Siddhartha Engineering College** (Autonomous under JNTUK) during the year 2020-21.

(Dr. Shaik. Fathimabi)
Sr. Assistant Professor
Dept. of Information Technology

(Dr. M. Suneetha)
Professor &Head
Dept. of Information Technology

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

First and foremost, I sincerely salute our esteemed institution V.R SIDDHARTHA ENGINEERING COLLEGE for giving me this opportunity for fulfilling my MAJOR project. I am grateful to our principal Dr. A.V. RATNA PRASAD, for his encouragement and support all through the way of my project.

On the submission of this major project report, I would like to extend my honour to Dr. M. Suneetha, Head of the Department, IT for her constant motivation and support during my work.

I feel glad to express my deep sense of gratefulness to my project guide Dr. S. Fathimabi, Sr. Assistant Professor for Her guidance and assistance in completing this project successfully.

I would also like to convey my sincere indebtedness to all faculty members, including supporting staff of the Department, friends, and family members who bestowed their great effort and guidance at appropriate times without which it would have been very difficult on my part to finish the project work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

S.No	Equation	Description
1	$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}$	Equation to calculate Term Frequency
2	$w_{i,j} = tf_{i,j} \text{ x } \log(\frac{N}{df_i})$	TF-IDF - product of term frequency and inverse document frequency
3	$F_0(x) = \arg\min \sum_{i=1}^{n} \mathrm{L}(y_i, \gamma)$	Initializing the base model with the mode of labels. Such that the loss is minimum,
4	$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$	Equation to calculate the pseudo residuals
5	$\gamma_m = arg_\gamma min \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))$	Solving the one dimensional problem to calculate the multiplier γ
6	$F(x) = F_{m-1}(x) + \gamma h_m(x_i)$	Updating XGBoost model based on previous model and multiplier γ
7	$O_i = \sigma(W_i x_i + b_i)$	Activation function σ applied to the weighted sum of inputs and bias
8	$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$	Accuracy is the ratio of count of correct predictions to the total no of predictions.
9	$Precision = \frac{TP}{TP + FP}$	Precision is the ratio of correctly predicted positive observations to the total predicted positive observations
10	$Recall = \frac{TP}{TP + FN}$	Recall is the ratio of correctly predicted positive observations to the all observations in actual class
11	$F1 \text{ score} = 2 \times \frac{Precision \times Recall}{Precision + Recall}$	F1 Score is the weighted average of Precision and Recall

ABSTRACT

Numerous initiatives have been developed to prevent suicide. Still, people are not seeking help considering it will dishonor them in society, and they are unwilling to reveal their identity. However, thanks to social media platforms, people are more willing to express themselves anonymously as they don't directly face anyone. This work aims to use this vast data to identify texts containing suicidal thoughts and help early detection of suicide using machine learning and deep learning techniques. The dataset used is collected by taking posts from "SuicideWatch" and "teenagers" Subreddits of Reddit using, "Pushshift" API. The proposed model uses a Universal Sentence Encoder to encode the text and a Fully connected Neural Network(FCNN) to classify text into suicide, and non-suicide, which outperformed other baseline models with an accuracy of 94.16 %.

Keywords: Early suicide detection, SuicideWatch, teenagers, Subreddit, Reddit, Pushshift, Universal Sentence Encoder, Fully Connected Neural Network.

CHAPTER-1

INTRODUCTION

Every project arrives from the problems that are facing by society. Here in this chapter, the motivation of taking the project, the background behind it, the problem facing, and the real-time application are presented.

Each year, close to Eighty lakh people commit suicide worldwide, one person for every 40 seconds. In low, and middle-income countries, 79% of suicides were reported. In the age group of 15-39, suicide is noted as one of the significant reasons for death [1]. Suicide ideation can be viewed as a propensity to end one's life [2]. People may commit suicide for many reasons: It can be personal, financial, etc. These people can either be called suicide ideators (or) suicide attempters [3].

In recent years, the practice of social media has been exploding. There are many social media platforms existing today, and people are very keen on using these platforms. Some social media platforms are private messaging applications, while some social media platforms are forums where people post about their daily activities and updates about their personal lives. People are highly interested in publishing their daily updates, and they keep on using them; this has been an addiction at present, and this has been at an extreme level.

Users in social media can be native or anonymous, where people tend to participate in different communities and express their opinions on a particular topic. With its mental health-related communities, social media provides valuable research platforms for the development of new methodologies. Furthermore, it offers an opportunity to detect suicide ideation in social media forums. Deep learning algorithms are proven impressive in many computer vision and pattern recognition problems. They use neural networks that provide better results to complex problems than traditional machine learning algorithms, which consume more time and are incomplete than deep learning. Deep learning can also detect suicide ideation in social media forums [4].

This work aims to provide valuable information related to suicide ideation available on social media platforms like Reddit by analyzing data and comparing the performance of different techniques. The main task is to try other word embedding techniques [5, 6] and test the performance of the Universal Sentence Encoder. In addition, This work will try to demonstrate that the Universal Sentence Encoder can outperform other embedding texts when used with a Fully Connected Neural Network [7].

The main intention of this study is to develop an effective method to detect suicidal content in social media platforms and compare the performance of the proposed method with other popular techniques. In this work, firstly, data from Reddit will be extracted and pre-processed. Then detection of the presence of suicidal ideation is done by computing the frequency of unigrams in the dataset. Finally, comparison of the proposed method USE-FCNN with other baseline models is shown.

This study has the following contributions:

- **Dataset:** This study will introduce a new dataset prepared by collecting posts from "SuicideWatch" and "depression" subreddits of the Reddit platform.
- **Unigram analysis:** unigram analysis shows that suicide-related conversations are discussed in the "SuicideWatch" subreddit and other topics in the "teenagers" subreddit.
- **Feature analysis:** Evaluation of performance of word embeddings, TF-IDF, bag-of-words over Universal Sentence Encoder.
- **Comparative evaluation:** This part of study compares the performance of the proposed USE-FCNN method and compare its performance with deep learning methods like LSTM, LSTM-CNN, and traditional machine learning models like SVM and XGBoost.

1.1. Origin of the problem

The social stigma associated with mental disorders deters patients from seeking help or sharing their experiences directly with others including clinicians. Nowadays people are opening about suicidal thoughts including depression on social media platforms as they can be anonymous. The objective of this work is to develop an efficient system capable of detecting suicidal thoughts present on social media platforms to help in the early detection of suicides.

1.2. Basic definitions and Background

A. Reddit

Reddit is a social news aggregation, web content rating, and discussion website, recently including live stream content through Reddit Public Access Network. Registered members submit content to the site such as links, text posts, and images, which are then voted up or down by other members. Posts are organized by subject into user-created boards called "communities" or "subreddits", which cover a variety of topics such as news, politics, science, movies, video games, music, books, sports, fitness, cooking, pets, and image-sharing.

B. Twitter

Twitter is an American micro-blogging and social networking service on which users post and interact with messages known as "tweets". Registered users can post, like, and re-tweet tweets, but unregistered users can only read them.

C. XGBOOST

XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. In prediction problems involving unstructured data (images, text, etc.) artificial neural networks tend to outperform all other algorithms or frameworks.

D. Universal Sentence Encoder

Universal sentence encoder (USE) encodes text into high dimensional vectors used for text classification, semantic similarity, clustering, and other natural language tasks. The model is trained and optimized for greater-than-word length text, such as sentences, phrases, or short paragraphs. The input is a variable length English text and the output is 512- dimensional vector.

E. Real-time

Real-time or real-time describes various operations in computing or other processes that must guarantee response times within a specified time (deadline), usually a relatively short time. Here real-time refers to the detecting of suicidal content in real-time using spark streaming.

F. Pushshift

Pushshift.io is a Reddit API to access posts, comments from subreddits of Reddit. It is a URL-based API that returns data in JSON format. The URL consists of the details regarding the subreddit from which the posts must be collected and returns posts that were created during a particular time interval which is also mentioned in the URL.

G. LSTM-CNN

This study uses LSTM and LSTM-CNN models pre-trained on 32-dimensional word2vec techniques to compare the proposed method. A time-step value of 600 is used. Thirty-two filters and a kernel size of 3 are used for the convolution layer, while the lstm layer has a dropout of 0.3 and a recurrent dropout of 0.3 to avoid over-fitting. Sigmoid is used as activation for the final layer, while relu is used for other layers. Adam optimizer is used to reduce the binary cross-entropy loss, and the models were trained over 20 epochs with a batch size of 256 with early-stopping with the patience of 5.

H. SVM

SVM is a popular algorithm that is widely used in text classification tasks [26, 27]. In addition, the supervised learning algorithm is known to perform well in mental health tasks [28]. This experiment implements SVM by providing the Universal Sentence Encoder output and the output of TF-IDF followed by CountVectorizer. SVM constructs hyper-plane in high-dimensional space to solve non-linearly and linearly separable problems in lower-dimensional space.

I. Fully Connected Neural Network

Fully connected neural networks (FCNNs) are a type of artificial neural network where the architecture is such that all the nodes, or neurons, in one layer are connected to the neurons in the next layer. While this type of algorithm is commonly applied to some types of data, in practice this type of network has some issues in terms of image recognition and classification. Such networks are computationally intense and may be prone to over-fitting. When such networks are also 'deep' (meaning there are many layers of nodes or neurons) they can be particularly hard for humans to understand.

Other internal abbreviations used are:

USE: Universal Sentence Encoder

FCNN: Fully Connected Neural Network

SVM: Support Vector Machines

LSTM: Long Short Term Memory

CNN: Convolutional Neural Networks

CV: Count Vectorizer

TF-IDF: Term Frequency Inverse Document Frequency

1.3. Problem Statement

Many posts associated with suicidal thoughts are being unnoticed in social media forums. These must be identified in real-time as the users who are having suicidal thoughts are at potential risk. India accounts for 17.8% of all reported suicide cases[1]. Online discussion forums have become a place for people to express suicidial ideas and this can be used to detect early suicidal thoughts and prevent them. The objective of this project is to develop efficient methods to identify suicidal and depression thoughts in social media forums and to use them in detection of depression and suicidal feelings by using historical data to identify suicidal and depression texts and to use spark streaming to identify suicidal contents in real-time.

1.4. Real-time Applications of Proposed work

- The project can be merged with popular social media platforms to identify posts carrying potential suicide thoughts.
- Given the user details, it can identify potential people having a risk of suicide and can be extended to provide support
- The application is also capable of identifying depression thoughts.

In this chapter, the problem of how suicidal posts are going unnoticed in social media is discussed, While specifying how the developed application can be used to overcome it.

CHAPTER-2

REVIEW OF LITERATURE

As the world demands updating itself to a new era there are many advancements in technology, the way problems are solved and implementing less cost computation is increasing day by day. In the process of understanding existing solutions, learning the experiences and building a solution that can satisfy real-time needs are important. In this section, previous papers are clearly studied and stated.

2.1. Description Of Existing Systems

Many techniques have been developed under suicide ideation in recent years, and most of these are based on machine learning and deep learning. In the study by Tadesse et al. [8], the analysis of suicide ideation is done using n-gram analysis, LSTM-CNN model, and other machine learning algorithms. The dataset contained both suicidal and non-suicidal posts and was collected from Reddit Social media from the subreddit r/SuicideWatch. It uses 10-fold cross-validation for hyper-parameter tuning and for the detection of suicide ideation in social media. The proposed model uses two frameworks. The model comprises a combination of CNN and LSTM characteristics and forms an LSTM-CNN model to classify text data. The architecture consists of a word embedding layer capable of developing fixed-length vectors followed by a dropout layer used to avoid over-fitting, followed by an LSTM layer to perform feature extraction, and the next pooling layer, flatten layer SoftMax function completes the neural network. Evaluation metrics like F1, Recall, Accuracy, and Precision are used to evaluate the baseline model for the deep learning classification technique.

Suicide is depicted as one of the most critical mental health issues. Having it detected at an early age can be especially useful for a person suffering. People express themselves on online platforms rather than directly to a person. So having their content monitored can be beneficial for the person. Shaoxiong et al. [9] used supervised learning to watch the person's content online, keeping analysis of a person's language choices while topic descriptions are used to detect a person's mental state. The sentiments from the post are also considered, i.e., whether it is

positive or negative. To identify suicide ideation, they have used four supervised classifiers and two neural networks. They have also included feature extractions like statistical, syntactical, linguistic, word embedding, topic features comparing six classifiers.

Conditions that may increase suicide risk were discovered by American Foundation for Suicide Prevention (AFSP). Vioules et al. [10] proposed that these can be used to identify an individual's risk by analyzing an individual's online behavior on Twitter by a series of observations over time. The aim is to detect a behavior change, and a feature that has been created for text analysis is known as SPA (Suicide Prevention Assistant). A Martingale framework has been developed for emotion change detection and was set on real-time data. Real-time data from Twitter was collected using the Twitter streaming API. Data has been classified into two classes: a) Distressed and b) Non-distressed. They have annotated tweets for four categories based on severity. To compare different algorithms, they have used 10-fold cross-validation.

Research by Kumar et al. [11] shows the connection between an increase in suicide following the death of a celebrity known as the Werther effect. Werther effect describes that followed by a celebrity suicide, there will be an increase in attempted or completed suicides. They have compiled celebrity suicides from Wikipedia. Other online social media platforms from which the discussion on celebrity suicide is collected, like Reddit/Twitter, are used more for discussions. There is a subreddit called "SuicideWatch," from which the dataset is collected. Their work helps identify the people who are very much influenced by their beloved celebrity's death and are in danger. The methods used here are measuring the post volume, linguistic measures like LIWC, lexical density, n-gram analysis, topic model analysis. They limited their work to the users of "SuicideWatch" only.

OSN's (Online social media networking) became a part of everyone's lives. Social media mining mainly focuses on discovering the behind-scenes of data for improvements. For example, Shuai et al. [12] developed an ML Framework for detecting SNMD's (Social networking Mental Disorder) called SNMDD (Social networking Mental Disorder Detection). 3126 OSN users worldwide were recruited by the researchers via MTurk to obtain data for training, and building classifiers in SNMDD and some social interaction features like SS, SDiv, and Parasocial

relationship, and proposed multi-source learning with tensor decomposition acceleration method to detect suicidal content.

Orthodox suicide ideation requires communication between doctors and patients. Doctors assess mental health by a set of questionnaires, and the answers can identify the feelings of the patient's questionnaire. The Global-level shortage for mental health care facilities, decrease in the number of doctors, stigma, neglect of mentally weak people, and the increase of internet and social media activity builds a bridge of communication between patients and doctors. Huang et al. [13] developed a method to detect suicide ideation based on multi-weighted fusion and extracted a linguistic feature set. They proposed a weak classification model that collaborated with feature sets to see results. Two algorithms SVM, and Decision tree, are used for classification. Based on the MFWF method, they achieved better performances than the data-driven method. Recall increased from 0.78 to 0.88 by 12.8%, but it is limited to a single social media site, single blogs, and they did not include other feature selection methods.

Real-time suicide ideation analysis on Twitter was proposed by Vioules et al. [31], who presented a framework by introducing a real-time detection problem. They defined online proxy measurements (behaviour features) for suicide warning signs. Following the text classification, two approaches are proposed, an NLP-based method that combines elements generated from text based on lexicons. The second approach is called a distress classifier based on machine learning. The NLP text-scoring approach successfully separates tweets exhibiting distress-related content and acts as a robust input into the martingale framework. However, the features used to determine the mental health of a user may differ between individuals. Therefore, it is tested only on two users. Spotting out regular language patterns in social media platforms follows through a significant suicide ideation detection in forums by applying numerous machine learning techniques on various Natural Language Processing techniques. Desmet et al. [32] built a suicide note analysis to identify suicide ideation using SVM classifiers to detect emotion. They used 15 binary classifiers, so there are 15*15 =255 possible combinations. Bootstrap resampling was used to determine the maximisation for every classifier used.

O'Dea et al. [33] tried to detect the level of concern among suicide-related posts using automatic machine learning classifiers(Linear Regression, SVM) on TF-IDF and human codes.

14% of suicide-related tweets were judged ‘strongly concerning’ and believed to warrant further investigation. Both human coders and automated computer classifiers achieved the same accuracy. Wang et al. [34] observed that CNN neural networks with layers like Conventional, non-pooling, and pooling layers outspread many NLP tasks and proved to give excellent results than orthodox NLP methods. It emphasises local n-gram features and prevents long-range interactions. Wang used Sentence Classification, Machine learning, Deep learning, and a Hybrid framework combining ML and DL, and researchers used pre-trained word vectors for evaluation.

Wang et al. [35] proposed deep learning and machine learning models to identify the persons who are about to commit suicide within 30 days to 6 months via the social activities of that person. They created and extracted three sets of features to detect suicidal persons in social media, who typically had posts mentioning pronouns related to suicide.

STATENet, a time-aware primary transformer-based model for preliminary screening of suicidal risk in social media, was proposed by Sawhney et al. [36]. They have created various practical aspects of STATENet for suicide ideation detection and aimed to detect the suicidal posts by priority. In addition, they quantified the features of STATENet by learning different emotional features.

2.2. Overall Summary of Literature Survey

Above referred papers proposed different methods to detect suicidal ideation in online user content. Most of them are implemented using Machine Learning and Deep learning algorithms while exploring relational and behavioural analysis. Most of the papers used ‘r/SuicideWatch’ [15] data which is a subreddit for users to express suicidal thoughts. The proposed methods doesn’t involve identification of suicidal thoughts with state-of-the-art text encoders. This paper proposes USE-FCNN consisting of Universal Sentence Encoder as an encoding technique and a Fully Connected Neural Network to perform classification which outperforms other state-of-the-art algorithms proposed previously such as XGBoost, LSTM-CNN models.

CHAPTER-3

PROPOSED METHOD

Dataset was balanced by under-sampling the normal-conversation labeled posts, also rows containing duplicate values and NULL values were removed. The accuracies of different models were compared to find the best model suitable for classification. The proposed USE-FCNN consisting of Universal Sentence Encoder as an encoding technique and a Fully Connected Neural Network to perform classification outperformed baseline models like SVM, XGBoost, LSTM and LSTM-CNN. The model was later deployed using Flask to predict suicide using a Web User Interface.

3.1. Design Methodology:

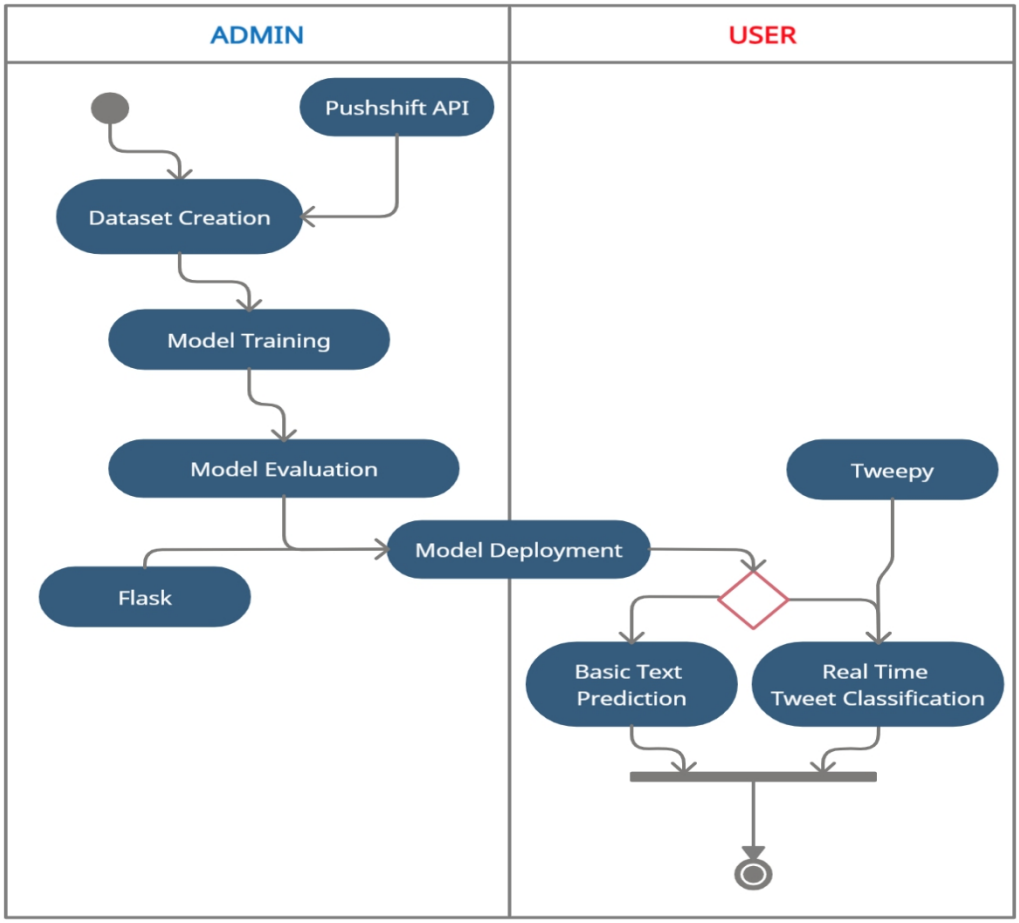


Fig 3.1 Activity Diagram

The Activity diagram of the problem solution is shown in Fig 3.1 The responsibility of the Admin/Developer includes collecting the dataset using Pushshift API. Later the data is given as input to different algorithms like XGBoost, SVM, LSTM, LSTM-CNN, and FCNN. Models were evaluated based on their accuracy scores and the best performing model i.e USE-FCNN is used for deployment. Deployment is done using Flask. Flask is a python library that can be used to deploy machine learning models to websites. Users can use the deployed model in two ways 1) Give a sentence/paragraph as input to the deployed model using web UI and predict the label for it. 2) Enter a keyword to extract 100 latest tweets from Twitter in real-time and classify them into suicide and depression.

3.2. System Architecture

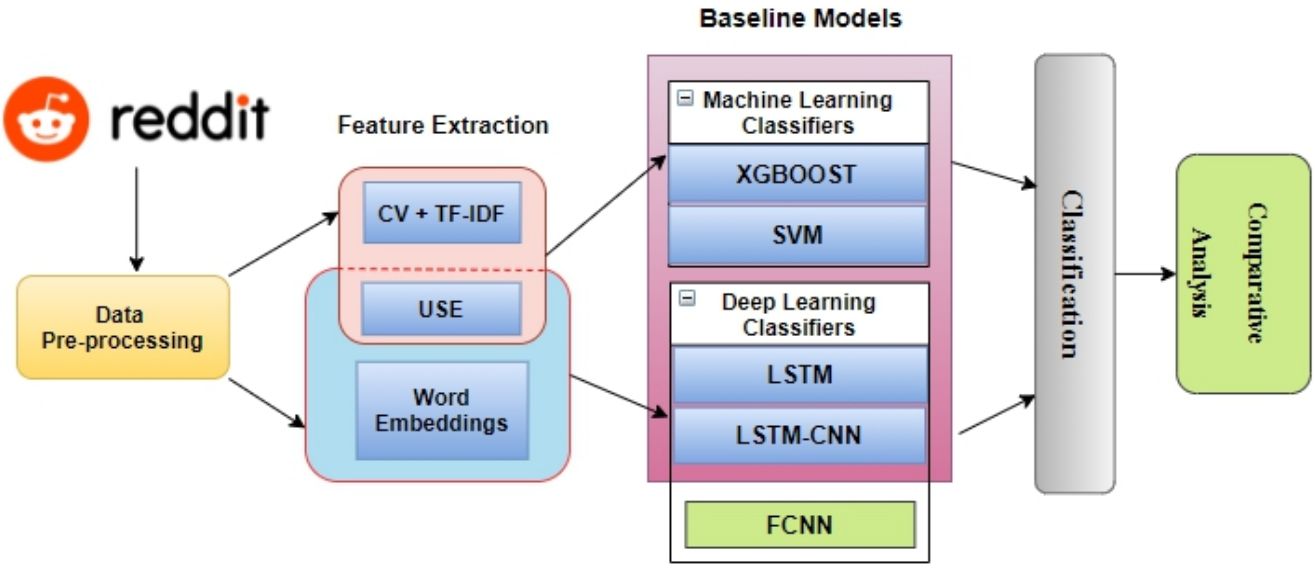


Fig 3.2 Architecture Diagram

Fig 3.2 shows the architecture of the proposed system consisting of Data Pre-processing, Feature Extraction, Algorithms being used, and the extraction of tweets from Twitter.

3.2.1. Data Pre-processing

Text data usually contains noise/redundant data that must be cleaned. Data pre-processing involves removing this noise contained in the form of special characters, numbers e.t.c. The URL addresses and multiple white spaces are replaced with a single white space using regular expressions. Dataset is also converted into a balanced dataset with the help of under-sampling. Under-sampling involves the removal of rows that are related to the class having several rows compared to another. Under-sampling ensures that both class labels have the same number of rows. The study uses the Natural Language Tool Kit (NLTK) [15] package of python to remove stop-words in the text data. Finally, the entire text data is converted into lower case characters before the usage of word embeddings.

3.2.2. TF-IDF

TF-IDF (Term frequency-inverse document frequency) was invented for document search and information retrieval. It works by increasing proportionally to the number of times a word appears in a document but is offset by the number of documents that contain the word. Equation 1 represents the term frequency, While equation 2 represents the product of term frequency and document frequency.

Consider an example with 2 documents/sentences:

1. 'The man went out for a walk'
2. 'The children sat around the fire'

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}} \quad (1)$$

$$w_{i,j} = tf_{i,j} \text{ x } \log(\frac{N}{df_i}) \quad (2)$$

The TF-IDF value of the word 'man' is 1/7 * (log(2/1)), While it is 1/7 * (log(2/2)) for the word 'the'.

3.2.3. Word embeddings

Word embeddings are a type of word representation that allows words with similar meanings to have a similar representation. They are a distributed representation for a text that is perhaps one

of the key breakthroughs for the impressive performance of deep learning methods on challenging natural language processing problems.

3.2.4. Count-Vectorizer

Many machine learning, and deep-learning algorithms do not work well with text data; hence they must be converted to numerical data in the form of numerical vectors. Count-Vectorizer is used in the creation of numerical vectors from text data. It creates columns equal to the unique number of words in the vocabulary. Then, each row is represented by using these columns, words are replaced by the frequency of words in that row.

3.3. Description of Algorithms

3.3.1. XGBoost

XGBoost is an open-source software library that provides a gradient boosting framework for C++, Java, Python, R, Julia, Perl, and Scala. It works on Linux, Windows, and macOS. The project description aims to provide a "Scalable, Portable and Distributed Gradient Boosting (GBM, GBRT, GBDT) Library". It runs on a single machine, as well as the distributed processing frameworks Apache Hadoop, Apache Spark, and Apache Flink. It has gained much popularity and attention recently as the algorithm of choice for many winning teams of machine learning competitions.

The algorithm differentiates itself in the following ways:

- Languages: Supports all major programming languages including C++, Python, R, Java, Scala, and Julia.
- Cloud Integration: Supports AWS, Azure, and Yarn clusters and works well with Flink, Spark, and other ecosystems.
- A wide range of applications: Can be used to solve regression, classification, ranking, and user-defined prediction problems.
- Portability: Runs smoothly on Windows, Linux, and OS X

XGBoost Algorithm

Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function *L(y, F(x))*, number of iterations *M*.

1) Initialize model with a constant value:

$$\boldsymbol{F_0(x)} \ = \mathbf{arg\,min} \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \mathbf{L}(\boldsymbol{y_{i,}}\, \boldsymbol{\gamma}) \tag{3}$$

2) For $m = 1$ to M:

i) Compute so-called pseudo-residuals:

$$\boldsymbol{r_{im}} = \ -\left[\frac{\boldsymbol{\partial L(y_i, F(x_i))}}{\boldsymbol{\partial F(x_i)}}\right]_{\boldsymbol{F(x) = F_{m-1}(x)}} \tag{4}$$

ii) Fit a base learner(or weak learner,e.g.tree) $h_m(x)$ to pseudo-residuals, i.e train it using the training set $\{(\mathrm{x_i}, \mathrm{y_i})\}_{\mathrm{i=1}}^{\mathrm{n}}$

iii) Compute multiplier γ_m by solving the following one-dimensional problem:

$$\boldsymbol{\gamma_m = arg_{\gamma} min} \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{L(y_i, F_{m-1}(x_i) \ + \gamma h_m(x_i))} \tag{5}$$

iv) Update the model:

$$\boldsymbol{F(x)} \ = \ \boldsymbol{F_{m-1}(x)} \ + \boldsymbol{\gamma h_m(x_i)} \tag{6}$$

3) Output $F_M(x)$

3.3.2. Universal Sentence Encoder

The Universal Sentence Encoder encodes text into high-dimensional vectors that can be used for text classification, semantic similarity, clustering, and other natural language tasks. The model is trained and optimized for greater-than-word length text, such as sentences, phrases, or short paragraphs. It is trained on a variety of data sources and a variety of tasks with the aim of dynamically accommodating a wide variety of natural language understanding tasks.

Few use cases of Universal Sentence Encoder are:

- As an embedding layer at the start of a deep learning model.
- Performing classification by finding semantically similar sentences.
- Getting rid of duplicate sentences or phrases before analysis.

There are two Universal Sentence Encoders to choose from with different encoder architectures to achieve distinct design goals, one based on the transformer architecture targets high accuracy at the cost of greater model complexity and resource consumption. The other targets efficient inference with slightly reduced accuracy by the deep averaging network (DAN). The encoder part of the original transformer architecture is used in this study. The architecture consists of 6 stacked transformer layers. Each layer has a self-attention module followed by a feed-forward network.

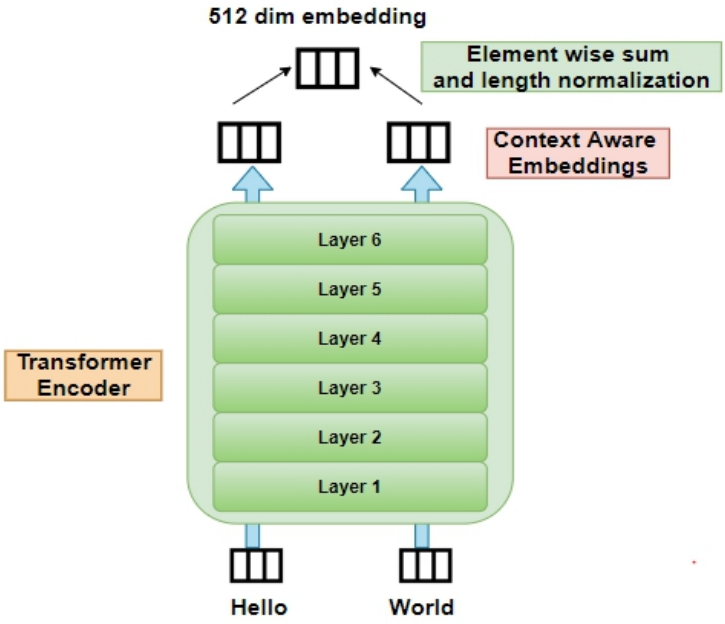


Fig 3.3 Architecture of USE

3.3.3. Gradient Descent

Gradient descent is an optimization algorithm that's used when training a machine learning model. It's based on a convex function and tweaks its parameters iteratively to minimize a given function to its local minimum. Gradient descent is simply used to find the values of a function's parameters (coefficients) that minimize a cost function as far as possible. It measures the change in all weights with regard to the change in error. You can also think of a gradient as the slope of a function. The higher the gradient, the steeper the slope, and the faster a model can learn. But if the slope is zero, the model stops learning. In mathematical terms, a gradient is a partial derivative with respect to its inputs.

Algorithm

1. Initialize the weights (a & b) with random values and calculate Error (SSE).
2. Calculate the gradient i.e., change in SSE when the weights (a & b) are changed by an exceedingly small value from their original randomly initialized value. This helps us move the values of a & b in the direction in which SSE is minimized.
3. Adjust the weights with the gradients to reach the optimal values where SSE is minimized
4. Use the new weights for prediction and to calculate the new SSE.
5. Repeat steps 2 and 3 till further adjustments to weights don’t significantly reduce the Error.

3.3.4. LSTM-CNN

Long Short Term Memory Network is an advanced RNN, a sequential network, that allows information to persist. It is capable of handling the vanishing gradient problem faced by RNN. A recurrent neural network is also known as RNN is used for persistent memory. Let’s say while watching a video you remember the previous scene or while reading a book you know what happened in the earlier chapter. Similarly RNNs work, they remember the previous information and use it for processing the current input. The shortcoming of RNN is, they cannot remember Long term dependencies due to vanishing gradient. LSTMs are explicitly designed to avoid long-term dependency problems.

Convolutional neural networks are composed of multiple layers of artificial neurons. Artificial neurons, a rough imitation of their biological counterparts, are mathematical functions that calculate the weighted sum of multiple inputs and outputs an activation value.

The LSTM and LSTM-CNN models pre-trained on 32 dimensional word2vec [16] are used to compare performance with the proposed method. A time-step value of 600 is used. Thirty-two filters and a kernel size of 3 are used for the convolution layer, while the lstm layer has a dropout of 0.3 and a recurrent dropout of 0.3 to avoid over-fitting. Sigmoid is used as activation for the final layer, while relu is used for other layers. Adam optimizer is used to reduce the binary cross-entropy loss, and the models were trained over 20 epochs with a batch size of 256 with early-stopping with the patience of 5.

3.3.5. Support Vector Machines

SVM is a popular algorithm that is widely used in text classification tasks [26, 27]. In addition, the supervised learning algorithm is known to perform well in mental health tasks [28]. This experiment implements SVM by providing the Universal Sentence Encoder output and the output of TF-IDF followed by CountVectorizer. SVM constructs hyper-plane in high-dimensional space to solve non-linearly and linearly separable problems in lower-dimensional space.

SVM Algorithm

Inputs: Determine the various training and test data

Outputs: Determine the calculated accuracy.

1. W represents orientation & b represents position.

$$g(x_1) = w^T x_1 + b < 0 \; implies \; x1 \text{ belongs to } C1$$

$$g(x_1) = w^T x_1 + b > 0 \; implies \; x1 \text{ belongs to } C2$$

2. For all training samples in C1

   i. Start with initial values of w and b, if $w^T x_1 + b > 0$ then ok, else

   ii. Need to modify w and b so that $w^T x_1 + b > 0$.

3. The same process will be repeated for C2 with $w^T x_1 + b < 0$.

3.3.6. Fully Connected Neural Network

Figure 3.4 shows the proposed Fully Connected Neural Network Architecture consisting of various layers which are described below.

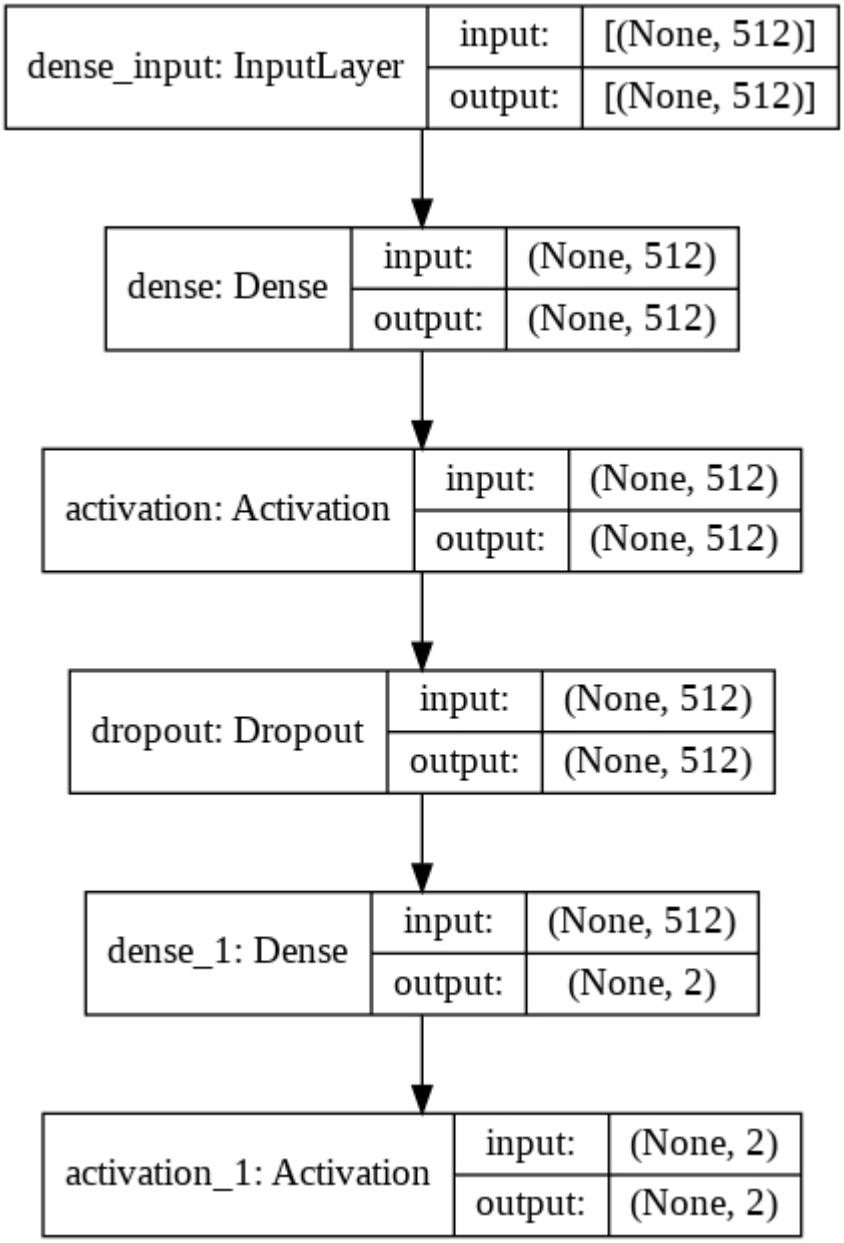
Dense Layer

The dense layer is one of the most commonly used neural network layers. A dense layer performs matrix multiplication of the input and the weights associated with the inputs, and bias can be added optionally. In addition, activation functions such as relu, leaky-relu, tanh can be applied over it. The unit parameter defines the number of neurons and the output vector of the dense layer. Equation 1 shows the mathematical representation of how a dense layer calculates the output. 'W' stands for weighted matrix while the 'x' is associated with the input matrix. 'b' denotes bias while" means the activation function.

$$O_i = \sigma(W_i x_i + b_i) \tag{7}$$

Dropout Layer

The purpose of the usage of the dropout layer in the model architecture is to overcome over-fitting. A 0.3 dropout rate is used in the proposed method. The dropout layer aims to randomly turn off or remove some neurons in the layer over which the dropout is applied [17]. Thus, dropout helps in randomly dropping noise in the training data. Dropout also helps in reducing training time besides avoiding over-fitting.



3.4 Architecture of Fully Connected Neural Network

Output Layer

The main motive behind using the output layer is to calculate the chances of the given text belonging to suicide and non-suicide text. The output layer receives input from the dense layer. The output layer can have two neurons with a softmax [18] activation function or a single

neuron with a sigmoid [19] activation function. The proposed model uses two neurons with a softmax [18] activation function.

Model Architecture and Parameters

The first step in the model architecture involves using the Universal Sentence Encoder based on transformer architecture. First, the Universal Sentence Encoder is used to encode texts into 512-dimensional vectors. Next, these 512-dimensional vectors are passed as input to an artificial neural network consisting of 1 hidden layer.

Table 3.1 represents the values of different parameters for the proposed method. The models are built using Keras, a deep learning framework.

Table 3.1 Parameter values for the proposed Fully Connected Neural Network

Layers	Parameters	Values
Dense layer	Units	512
	Activation Function	Relu
	Dropout	0.3
Output Layer, and others	Units	2
	Activation Function	SoftMax
	Batch size	32
	Epochs	20
	Optimizer	Adam
	loss	Binary cross-entropy

3.4. Description of datasets and tools

3.4.1. Dataset

The dataset is a collection of posts from 'SuicideWatch' [15], 'depression', and 'teenagers' subreddits of the Reddit platform. The data is collected using the Push-shift API. All

posts that were posted in SuicideWatch from Dec 16, 2008, till Jan 2, 2021, were collected while posts from ‘depression’ were collected from Jan 1, 2009, till Jan 2, 2021.

text	class
consists of title + body of posts in Reddit	Source of the post that can be used as target column
609811 unique values	SuicideWatch 50% depression 50% Other (40) 0%
Was going to hang myself but didn't have guts enough to kick away the chairI was all set to hang mys...	SuicideWatch
Have you ever maintained a poor friendship just to keep the last friend that you have?My friend and ...	depression

Fig 3.5 Dataset Description

3.4.2. Tweepy

Tweepy is a Python library that can be used to access the Twitter API [16]. Tweepy requires a Twitter developer account to access the Twitter API. Tweepy was used to collect a total of 50 latest posts based on a query keyword along with their time of creation. Also, it collects tweets that are written in English only.

3.4.3. Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a

scripting or glue language to connect existing components together. Python's simple, easy-to-learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms and can be freely distributed.

3.4.4. Keras

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.

3.4.5. Sklearn

Scikit-learn is probably the most useful library for machine learning in Python. The sklearn library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering, and dimensionality reduction. The scikit-learn project started as scikits.learn a Google Summer of Code project by David Cournapeau. Its name stems from the notion that it is a "SciKit" (SciPy Toolkit), a separately developed and distributed third-party extension to SciPy. The original codebase was later rewritten by other developers. In 2010 Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, and Vincent Michel, all from the French Institute for Research in Computer Science and Automation in Rocquencourt, France, took leadership of the project and made the first public release on February the 1st 2010. Of the various scikits, scikit-learn, as well as scikit-image were described as "well-maintained and popular" in November 2012. Scikit-learn is one of the most popular machine learning libraries on GitHub. Scikit-learn is largely written in Python and uses NumPy extensively for high-performance linear algebra and array operations. Furthermore, some core algorithms are written in Cython to improve performance. Support vector machines are implemented by a Cython wrapper around LIBSVM; logistic regression and linear support vector machines by a similar wrapper around LIBLINEAR. In such cases, extending these methods with Python may not be possible.

Scikit-learn integrates well with many other Python libraries, such as Matplotlib and Plotly for plotting, NumPy for array vectorization, Pandas data-frames, SciPy, and many more.

3.4.6. Flask

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies, and several common framework-related tools. Applications that use the Flask framework include Pinterest and LinkedIn.

3.4.7. Google Colab

Google Colaboratory (also known as Colab) is a free Jupyter notebook environment that runs in the cloud and stores its notebooks on Google Drive. Colab was originally an internal Google project; an attempt was made to open-source all the code and work more directly upstream, leading to the development of the "Open in Colab" Google Chrome extension, but this eventually ended, and Colab development continued internally. As of October 2019, the Colaboratory UI only allows for the creation of notebooks with Python 2 and Python 3 kernels; however, an existing notebook whose kernel spec is IR or Swift will also work, since both R and Swift are installed in the container. Julia language can also work on Colab (with e.g. Python and GPUs; Google's tensor processing units also work with Julia on Colab).

CHAPTER-4

RESULTS AND OBSERVATIONS

This section deals with performance of different algorithms on text classification in detail. The proposed method uses the USE-FCNN as it achieved better accuracy compared to baseline models like SVM, XGBoost, LSTM, and LSTM-CNN.

4.1 Stepwise Description of Results

A. *Creating the Dataset*

The dataset used is collected from Reddit, a social media platform that offers different communities known as Subreddits. Users can join any community of their choice, be anonymous, share, and discuss things related to the community via comments attached to a post [11]. The data is collected from 2 subreddits, 'SuicideWatch' and 'teenagers.' "SuicideWatch" is a subreddit where suicidal conversations were discussed, and it acts as peer support for anyone struggling with suicidal thoughts. While coming to "teenagers," its users can discuss anything related to teenagers, and not topic-specific like "SuicideWatch." The idea behind choosing "teenagers" is that it covers the most common teenager topics and can be used to identify non-suicidal posts. The data is collected from Reddit using the 'Pushshift' API. After performing data cleaning, it is made available on the Kaggle platform [14]. The title and body of posts are merged to form the text column. The dataset is balanced by removing duplicate rows and using under-sampling. It comprises 2,32,074 rows while each class comprises 1,16,037 rows.

The dataset is a collection of posts from 'SuicideWatch'[15], 'depression', and 'teenagers' subreddits of the Reddit platform. The data is collected using the Push-shift API. Push-shift API uses URLs to extract posts from a subreddit of Reddit along with title, body, creation time, etc. The dataset was uploaded to Kaggle and can be downloaded using the Kaggle package along with a kaggle_key and kaggle_username. Fig 4.1 shows how posts from Reddit are being extracted using Pushshift API.

```
data = getPushshiftData(1268811603, 1609520400, 'SuicideWatch')
data
```

```
https://api.pushshift.io/reddit/search/submission/?&size=1000&after=1268811603&before=1609520400&subreddit=SuicideWatch

[{'author': '[deleted]',
  'author_flair_background_color': '',
  'author_flair_css_class': None,
  'author_flair_text': None,
  'author_flair_text_color': 'dark',
  'brand_safe': False,
  'contest_mode': False,
  'created_utc': 1268866213,
  'domain': 'youtube.com',
  'full_link': 'https://www.reddit.com/r/SuicideWatch/comments/beqo4/a_song_for_the_day_cause_i_know_some_of_you_will/',
  'gilded': 0,
  'id': 'beqo4',
  'is_crosspostable': False,
  'is_reddit_media_domain': False,
  'is_self': False,
  'is_video': False,
  'link_flair_richtext': [],
  'link_flair_text_color': 'dark',
  'link_flair_type': 'text',
  'locked': False,
  'media_embed': {},
  'no_follow': False,
  'num_comments': 8,
  'num_crossposts': 0,
  'over_18': False,
  'parent_whitelist_status': 'no_ads',
```

Fig 4.1 Data Extraction using Pushshift API

The results are performed in two main phases. Firstly data analysis results are achieved, followed by classification analysis of models. The data analysis includes the generation of word clouds for suicidal and non-suicidal posts. Computation of frequencies of unigrams in both suicide and non-suicide-related posts is done and word clouds are created. The font size of words in the word cloud is based on their frequency, i.e., a term with high frequency is shown with large font in the word cloud. Figure 3 represents the word cloud of suicidal posts. Words like want, feel, life is displayed in large fonts since most of the suicide posts contain those words. We observe that the results are similar to findings of the suicide literature [29].

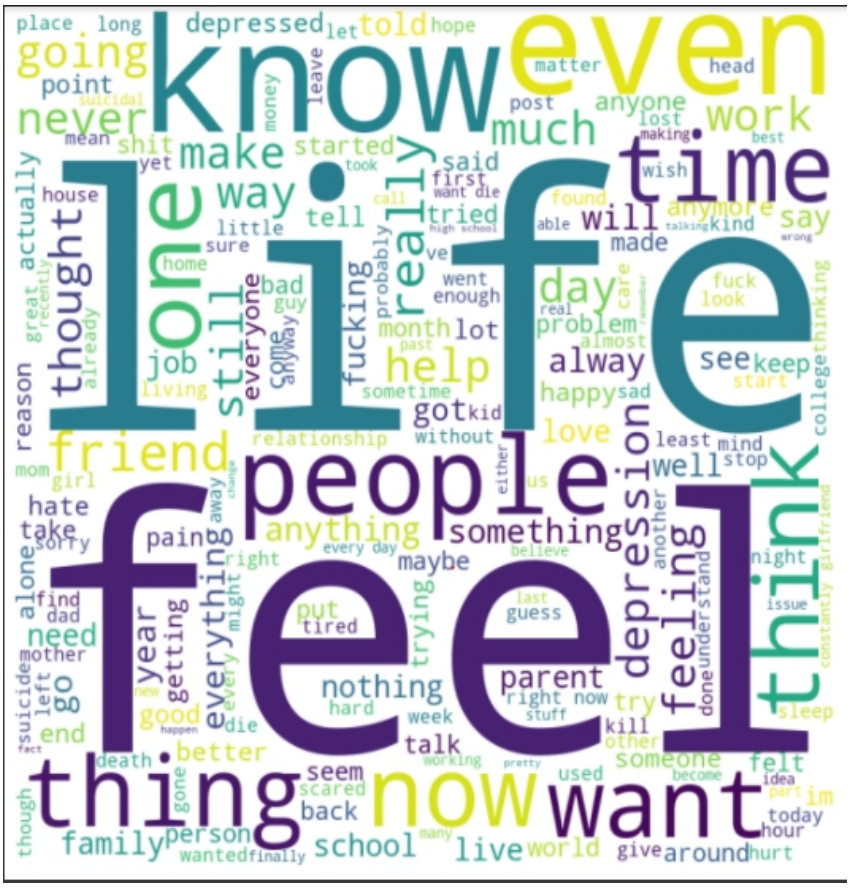


Fig 4.2 Word Cloud of Suicidal Posts

Fig 4.2 represents the word cloud of the suicidal posts. It describes the most used words for the classification purpose. The more the frequency of the words in the vocabulary the larger they appear in the word cloud. From Fig 4.2 we can observe the most frequent words that are mainly used for classification are ‘life’, ‘feel’ e.t.c.

On the other hand, the word cloud of non-suicidal posts had large fonts representing no highly repeated words in most non-suicidal posts. We can justify this as the data for non-suicidal posts is collected from the ‘teenagers’ subreddit, and users here talk on different topics; hence, there are no highly repeated words except for words such as ‘fuck’ are commonly used by teenagers. Figure 4.3 shows the word cloud for non-suicidal posts.

Figure 4.3 Word Cloud of Non-Suicidal posts

B. Training and comparing the performance of different algorithms

The baseline models SVM, XGBoost, LSTM, and LSTM-CNN along with the proposed USE-FCNN model are trained and evaluated to identify the best performing model. The performance of each algorithm is shown below with their classification reports.

4.1.1. XGBoost Performance Using TF-IDF + CV

Table 4.1 shows the classification report of the XGBoost algorithm having a feature type of combination of Term Frequency Inverse Document Frequency and Count-Vectorizer (TF-IDF + CV). We can observe that the precision value is low for the suicide label compared to the

Normal-Conversation label, While the recall value is low for the Normal-Conversation label compared to the Suicide label, The overall F1 score is less for the Normal-Conversation label.

Table 4.1 XGBoost with TF-IDF + CV Classification Report

Label	Precision	Recall	F1 score
Normal-Conversation	0.92	0.84	0.88
Suicide	0.86	0.93	0.89
Macro Average	89	88.5	88.7
Weighted Average	89	88.5	88.7

4.1.2. XGBoost Performance Using USE

Table 4.2 shows the classification report of the XGBoost algorithm having a feature type of the proposed Universal Sentence Encoder (USE). We can observe that the precision value is low for the suicide label compared to the Normal-Conversation label, While the recall value is low for the Normal-Conversation label compared to the Suicide label same as in the case of XGBoost with the feature type of TF-IDF + CV. The overall F1 score is the same for both labels.

Table 4.2 XGBoost with USE Classification Report

Label	Precision	Recall	F1 score
Normal-Conversation	0.94	0.93	0.93
Suicide	0.93	0.94	0.93
Macro Average	93.5	93.5	93.5
Weighted Average	93.5	93.5	93.5

4.1.3. SVM Performance Using TF-IDF + CV

Table 4.3 shows the classification report of the SVM algorithm having a feature type of combination of Term Frequency Inverse Document Frequency and Count-Vectorizer (TF-IDF + CV). We can observe that the precision value is low for the suicide label compared to the Normal-Conversation label, While the recall value is low for the Normal-Conversation label compared to the Suicide label, The overall F1 score is the same for both labels.

Table 4.3 SVM with TF-IDF + CV Classification Report

Label	Precision	Recall	F1 score
Normal-Conversation	0.93	0.92	0.93
Suicide	0.92	0.94	0.93
Macro Average	93	93	93
Weighted Average	93	93	93

4.1.4. SVM Performance Using USE

Table 4.4 shows the classification report of the SVM algorithm having a feature type of the proposed Universal Sentence Encoder (USE). We can observe that the precision value is the same as with the SVM using the TF-IDF + CV encoder for the Normal-Conversation label, While it increased to 0.95 for the Suicide label compared to the 0.92 precision of the SVM having TF-IDF + CV encoder. The recall value for the Normal-Conversation label increased to 0.95 compared to the SVM with the feature type of TF-IDF + CV having 0.92, While the recall of the Suicide label decreased from 0.94 to 0.93 compared to the SVM with TF-IDF + CV encoding. The overall F1 score is the same for both labels and is more than the previous SVM.

Table 4.4 SVM with USE Classification Report

Label	Precision	Recall	F1 score
Normal-Conversation	0.93	0.95	0.94
Suicide	0.95	0.93	0.94
Macro Average	94	94	94
Weighted Average	94	94	94

4.1.5. LSTM Performance Using Word Embeddings

Table 4.5 shows the classification report of the LSTM algorithm having a feature type of Word Embeddings implemented using Word2Vec trained of Google-News. We can observe that the precision value is low for the suicide label compared to the Normal-Conversation label, While the recall value is low for the Normal-Conversation label compared to the Suicide label The overall F1 score is less for the Normal-Conversation label.

Table 4.5 LSTM Performance Using Word Embeddings

Label	Precision	Recall	F1 score
Normal-Conversation	0.93	0.92	0.92
Suicide	0.92	0.93	0.92
Macro Average	92.5	92.5	92.5
Weighted Average	92.5	92.5	92.5

4.1.6. LSTM Performance Using USE

Table 4.6 shows the classification report of the LSTM algorithm having a feature type of the proposed Universal Sentence Encoder (USE). We can observe that the precision value is low for the suicide label compared to the Normal-Conversation label, While the recall value is low for the Normal-Conversation label compared to the Suicide label same as in the case of LSTM with the feature type of Word Embeddings. The overall F1 score is the same for both and a little higher for the Normal-Conversation label.

Table 4.6 LSTM Performance Using USE

Label	Precision	Recall	F1 score
Normal-Conversation	0.84	0.80	0.82
Suicide	0.79	0.83	0.81
Macro Average	81.5	81.5	81.5
Weighted Average	81.5	81.5	81.5

4.1.7. LSTM-CNN Performance Using Word Embeddings

Table 4.7 shows the classification report of the LSTM-CNN model having a feature type of Word Embeddings implemented using Word2Vec trained of Google-News. We can observe that the recall value is low for the suicide label compared to the Normal-Conversation label, While the precision value is low for the Normal-Conversation label compared to the Suicide label. The overall F1 score is the same for both the labels.

Table 4.7 LSTM-CNN Performance Using Word Embeddings

Label	Precision	Recall	F1 score
Normal-Conversation	0.91	0.94	0.93
Suicide	0.94	0.92	0.93
Macro Average	92.5	93	93
Weighted Average	92.5	93	93

4.1.8. LSTM-CNN Performance Using USE

Table 4.8 shows the classification report of the LSTM-CNN model having a feature type of the proposed Universal Sentence Encoder (USE). We can observe that the precision value is lower compared to the LSTM model for the Normal-Conversation label, While the precision for the suicide label increased to 0.88 compared to the 0.79 precision of the LSTM model. The recall value for the Normal-Conversation label increased to 0.86 compared to the LSTM with the feature type of USE having 0.80, While the recall of the Suicide label decreased from 0.83 to 0.80 compared to the LSTM with Universal Sentence Encoder encoding. The overall F1 score increased for both the labels compared to the LSTM with USE encoding, While the F1 score for Suicide was a little higher compared to the Normal-Conversation label.

Table 4.8 LSTM-CNN Performance Using USE

Label	Precision	Recall	F1 score
Normal-Conversation	0.79	0.86	0.82
Suicide	0.88	0.80	0.84
Macro Average	81.5	81.5	81.5
Weighted Average	81.5	81.5	81.5

4.1.9. FCNN Performance Using Word Embeddings

Table 4.9 shows the classification report of the proposed Fully Connected Neural Network model having a feature type of Word Embeddings implemented using Word2Vec trained of Google-News. We can observe that the precision value is low for the suicide label compared to the Normal-Conversation label, While the recall value is low for the Normal-Conversation label compared to the Suicide label. The overall F1 score is the same for both labels.

Table 4.9 FCNN Performance Using Word Embeddings

Label	Precision	Recall	F1 score
Normal-Conversation	0.93	0.89	0.91
Suicide	0.89	0.92	0.91
Macro Average	91	90.5	91
Weighted Average	91	90.5	91

4.1.10. FCNN Performance Using USE

Table 4.10 shows the classification report of the proposed FCNN-USE model having a feature type of the proposed Universal Sentence Encoder (USE) to the Fully Connected Neural Network. We can observe that the recall value is low for the suicide label compared to the Normal-Conversation label, While the precision value is low for the Normal-Conversation label compared to the Suicide label same as the LSTM-CNN model with the feature type of Word Embeddings. The overall F1 score is the same for both labels and is the highest among all the models.

Table 4.10 FCNN with USE Classification Report

Label	Precision	Recall	F1 score
Normal-Conversation	0.94	0.95	0.94
Suicide	0.95	0.94	0.94
Macro Average	94.5	94.5	94.5
Weighted Average	94.5	94.5	94.5

4.1.11. Evaluation Metrics

To evaluate the performance of different algorithms with the proposed text classifying method, we have used different evaluation metrics such as accuracy, Equation (8), and classification reports consisting of F1 score (F1), Equation (9), precision (P), and recall (R). Accuracy is the ratio of correct classifications to the total number of classifications, while the F1 score is the harmonic mean of precision and recall. Furthermore, precision helps find relevant results, while recall gives the portion of results correctly classified by the algorithm. Thus, a suitable algorithm will have good accuracy and an F1 score. The recall and precision must be close to achieve an excellent F1 score.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{8}$$

$$Precision = \frac{TP}{TP + FP} \tag{9}$$

$$Recall = \frac{TP}{TP + FN} \tag{10}$$

$$F1\ \text{score} = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{11}$$

TP = True Positive, TN = True Negative, FP = False Positive, FN = False Negative

4.1.12. Comparison of Performance of Algorithms

Table 4.11 shows the comparison of performances of various methods in terms of evaluation metrics. We can see that the proposed Fully Connected neural network using Universal Sentence Encoder's embeddings has outperformed other baseline models in terms of all evaluation metrics. The SVM implemented with Stochastic Gradient Descent has performed well next to the proposed method. Algorithms using USE as feature type have outperformed algorithms that are using CV+TF-IDF as their encoders. The LSTM, LSTM-CNN using Word2Vec as text encoder did not perform well compared to baseline models like SVM and XGBoost using USE. LSTM, LSTM-CNN performed very poorly when USE is used as an

encoder for them compared to Word2Vec. Also, the proposed FCNN using Universal Sentence Encoder as feature type has outperformed the FCNN implemented using Word2Vec.

Table 4.11 Performance of Classification Models

Methods	Feature Type	Accuracy	F1-Score	Recall	Precision
SVM(SGD)	CV + TF-IDF	92.8	93	93	93
	USE	93.8	94	94	94
XGBoost	CV + TF-IDF	88.6	88.7	88.5	89
	USE	93.3	93.5	93.5	93.5
LSTM	Word2vec	92.4	92.5	92.5	92.5
	USE	81.5	81.5	81.5	81.5
LSTM - CNN	Word2vec	92.6	93	93	92.5
	USE	83.18	83	83	83.5
FCNN	Word2vec	90.77	91	90.5	91
	USE	94.16	94.5	94.5	94.5

CNN = Convolutional Neural Networks, CV = Countvectorizer, FCNN = Fully Connected Neural Network, LSTM = Long Short Term Memory, SGD = Stochastic Gradient Descent, SVM = Support Vector Machine, USE = Universal Sentence Encoder, XGBoost = Extreme Gradient Boosting.

Figure 4.4 shows the graphical representation of the comparison of different algorithms with the proposed FCNN-USE model. The bar-graphs in green color represent algorithms using Universal Sentence Encoder as feature type, While the red color represents algorithms having the TF-IDF + CV as an encoder which is used for the SVM and XGBoost machine learning algorithms. The yellow-colored bar graphs represent algorithms having Word Embeddings as feature type. From the graph, we can observe that the algorithms having Universal Sentence Encoder as feature type produced better results compared to the same algorithms having the TF-IDF + CV encoder. While the deep learning models LSTM and LSTM-CNN achieved good

performance using the Word Embeddings as feature type they performed poorly with the Universal Sentence Embeddings it does not encode text at the word level. The proposed Fully Connected Neural Network achieved the best results when used with the Universal Sentence Encoder as an encoding technique instead of the Word Embeddings.

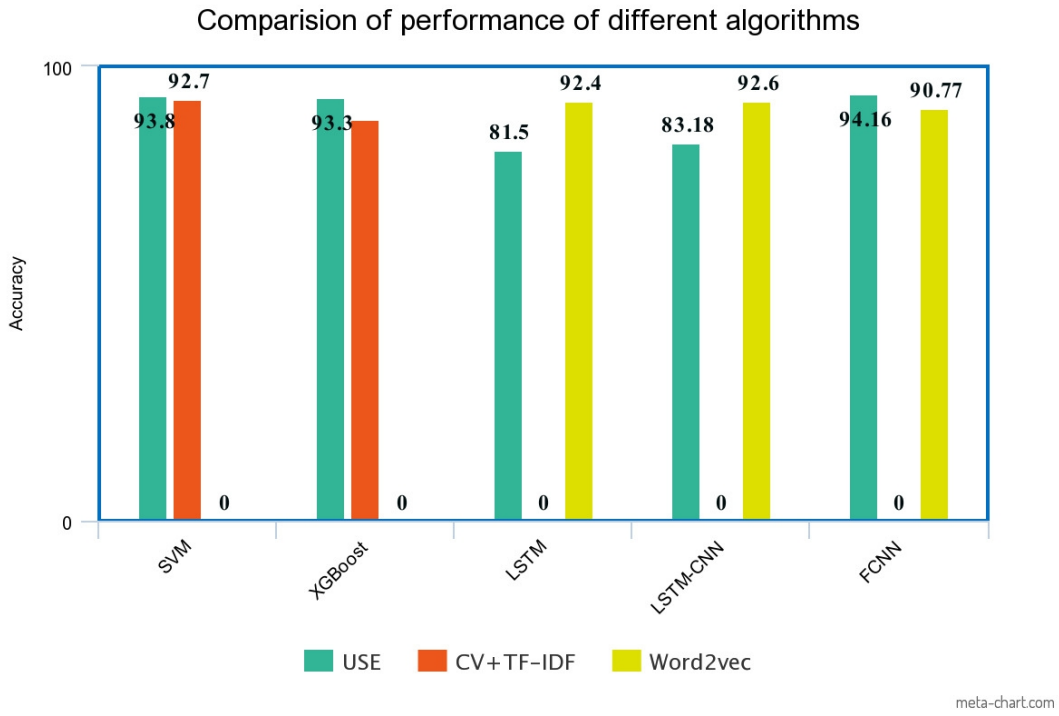


Fig 4.4 Comparison of Accuracies of Algorithms

Fig 4.4 shows a bar graph with algorithms on the x-axis and their accuracies on the y-axis. The proposed FCNN-USE model achieved the highest accuracy followed by SVM and XGBoost with the same encoding technique. While the LSTM and LSTM-CNN models with the USE encodings performed extremely poorly compared to the other models.

C. *Real-Time Extraction of Tweets*

Tweets can be extracted in real-time using the Tweepy API. Tweepy is an official Twitter API available for python to extract tweets based on keywords. Fig 4.5 shows how Tweepy is being authorized using the following requirements.

Requirements - Tweepy requires a Twitter developer account and having access to api_key, api_secret_key, access_token, access_token_secret. The API must be first authenticated using these four keys. The api.search() method is used to extract the tweets based on a keyword along

with the number of tweets that are required. The language of the tweets that are going to be extracted can be specified and only the latest tweets in that language are displayed.

```
api_key = "YfsfRUtv0Jstlvm0TLg8DrZNA"
# api secret key
api_secret_key = "Dx95SabGPVACrlQanwkajOnsfss0tWsyej8xO8rUKnf6N70Tyh"
# access token
access_token = "704330902432669696-pmTtYoAM3ywia3zAY5sWAEVkzhWUwan"
# access token secret
access_token_secret = "BSW1LmSmDZmNrDPL3KytWXgZeOTHo99Ee1vDu1FBc5EAJ"

authentication = tweepy.OAuthHandler(api_key, api_secret_key)
authentication.set_access_token(access_token, access_token_secret)
api = tweepy.API(authentication, wait_on_rate_limit=True)

def get_related_tweets(text_query):
    # list to store tweets
    tweets_list = []
    # no of tweets
    count = 50
    try:
        # Pulling individual tweets from query
        for tweet in api.search(q=text_query, count=count, lang = 'en'):
            print(tweet.text)
            # Adding to list that contains all tweets
            tweets_list.append({'created_at': tweet.created_at,
                                'tweet': tweet.text})
        return pd.DataFrame.from_dict(tweets_list)
```

Fig 4.5 Authorizing Tweepy

D. Deploying the Model

The model weights can be saved as a joblib file and can be used in deployment for prediction without having to train again. Flask was used to deploy the model to the local host. It can either extract real-time tweets from Twitter and classify them based on keywords or provide a single paragraph or sentence as input and predict whether it is associated with suicide or depression. Fig 4.6 shows the model being deployed running on a local host.

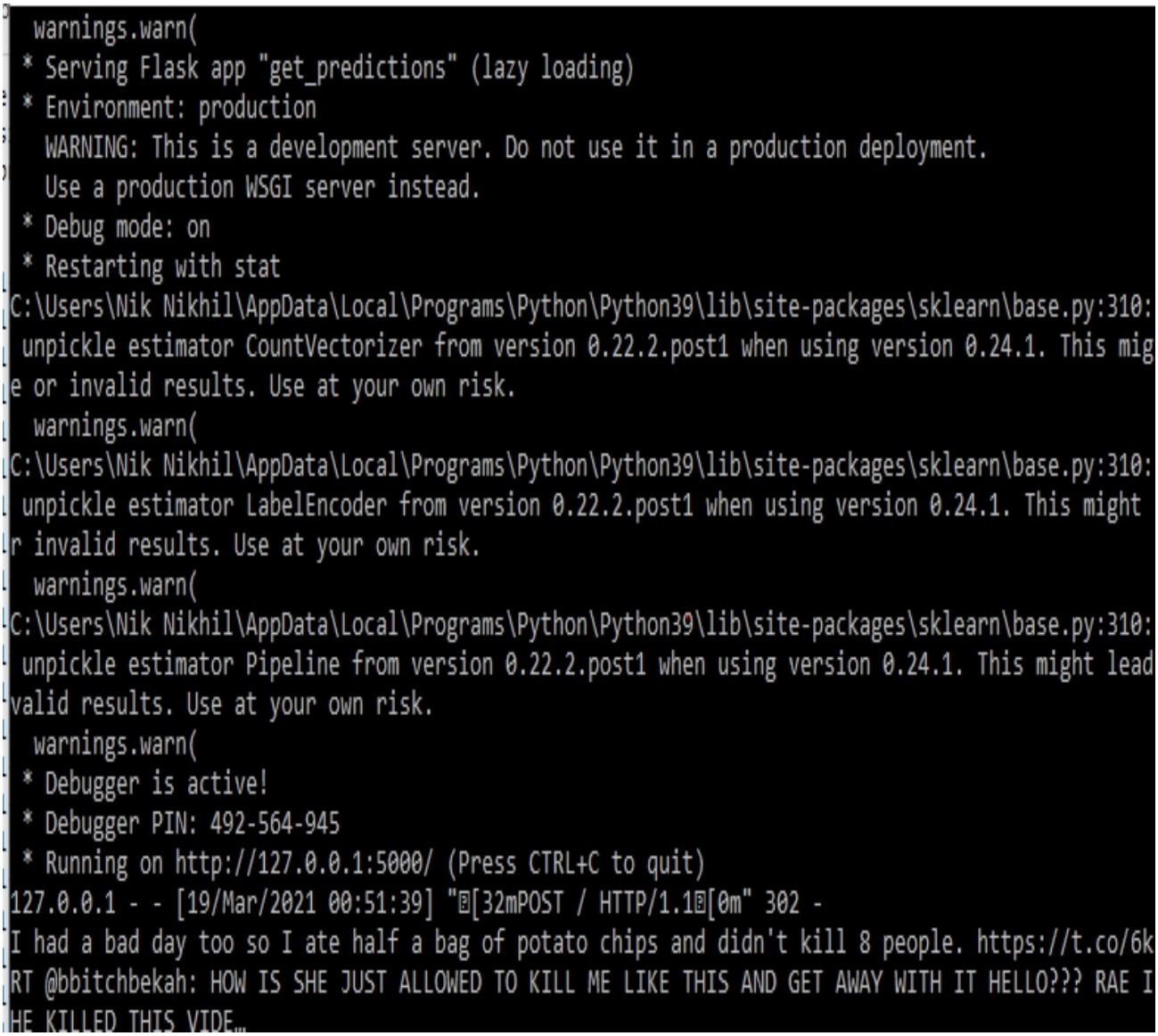


Fig 4.6 Model Deployment

4.2 Test Case Results

The following section discusses the test case results of executing the deployed Flask model

4.2.1 Extracting Tweets from Twitter

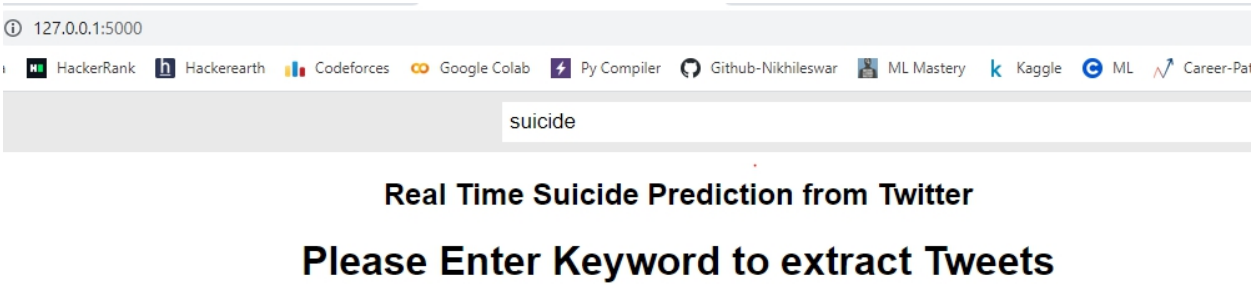


Fig 4.7 Tweets Extraction from Twitter

Fig 4.7 represents how tweets are going to be extracted based on the entered keyword from Twitter using a website user interface that is going to make use of 'Tweepy' package to extract tweets in real-time. Here tweets are collected based on the 'suicide' keyword. The Tweepy package is going to extract 100 latest real-time tweets from Twitter containing the 'suicide' keyword.

4.2.2 Classification of Tweets in Real-Time

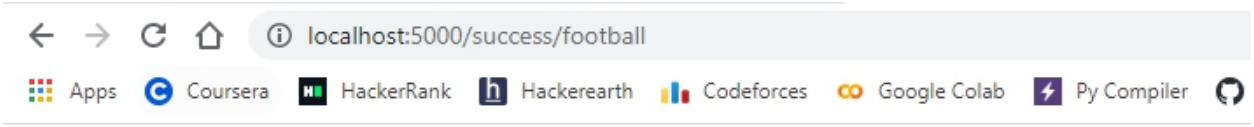
The proposed model was deployed using Flask and tested on real-time tweets collected from Twitter based on keywords. It can collect 100 latest tweets at a time from Twitter using the Tweepy API which is classified to identify suicide and depression-related tweets. The low count of depression and suicide class labels can be attributed to the fact that they are discussed in less number compared to normal conversations. Fig 4.8 shows tweets along with their text and predicted labels. It also shows the number of suicide and depression tweets identified along with the creation time of tweets. The above tweets are extracted for the "suicide" keyword which can be seen in Fig 4.7.

```
The entered Keyword is suicide
Normal_Conversation    64
Suicide                36
Name: prediction, dtype: int64

            created_at                                              tweet           prediction
0  2021-06-24 06:49:34  RT @bartell_harry: Suicide prevention isn't ju...              Suicide
1  2021-06-24 06:49:33  RT @NationBreaking: ANTI-VIRUS SOFTWARE entrep...  Normal_Conversation
2  2021-06-24 06:49:32  RT @Mick11656880: Is that the same @Jesse_Norm...  Normal_Conversation
3  2021-06-24 06:49:31  RT @_Kaushal_singh: #मोदीजी_शिक्षामित्रों_की_ब...  Normal_Conversation
4  2021-06-24 06:49:30  RT @officialmcafee: Getting subtle messages fr...  Normal_Conversation
..                 ...                                                ...                  ...
95 2021-06-24 06:47:16  RT @Chris__TF: Help is available. \nSuicide pr...              Suicide
96 2021-06-24 06:47:15  just remember #JohnMcAfee "committed suicide" ...  Normal_Conversation
97 2021-06-24 06:47:15  @TLA_lich It was a suicide. Just like Epstein....              Suicide
98 2021-06-24 06:47:13  Kureto told me about it. You realize this is a...              Suicide
99 2021-06-24 06:47:11  RT @officialmcafee: Getting subtle messages fr...  Normal_Conversation

[100 rows x 3 columns]
```

Fig 4.8 Tweets Classification in Real-Time - 1




```
The entered Keyword is football
Normal_Conversation    100
Name: prediction, dtype: int64

            created_at                                              tweet           prediction
0  2021-06-20 15:35:54  @ryanseymour1983 They might suddenly be very n...  Normal_Conversation
1  2021-06-20 15:35:54  RT @brfootball: Football's most iconic penalty...  Normal_Conversation
2  2021-06-20 15:35:54  RT @CryptoBlubber: ***VuVu Millionaire***\n\nE...  Normal_Conversation
3  2021-06-20 15:35:54  When I say I'm determined to present at the eu...  Normal_Conversation
4  2021-06-20 15:35:53  RT @ScottishFA: "If I didn't have football, I'...  Normal_Conversation
..                 ...                                                ...                  ...
95 2021-06-20 15:35:17  RT @Nigerianscamsss: Calling Jadon Sancho inex...  Normal_Conversation
96 2021-06-20 15:35:17  RT @6BDORS: Lionel Messi has dominated footbal...  Normal_Conversation
97 2021-06-20 15:35:16  it was a nigga on top playin football 😒 let it...  Normal_Conversation
98 2021-06-20 15:35:16  RT @CashKingLordJ: Big weekend of football for...  Normal_Conversation
99 2021-06-20 15:35:15  RT @6BDORS: Lionel Messi has dominated footbal...  Normal_Conversation

[100 rows x 3 columns]
```

Fig 4.9 Tweets Classification in Real-Time - 2

Figure 4.9. shows the extraction of tweets in real-time based on the 'football' keyword. The figure shows that all the tweets were identified as Normal-Conversations since most texts related to football are general conversations and are not related to suicide.

4.2.3 Classifying a Single Sentence

Fig 4.10 Sample Prediction-1

Figure 4.10 shows a sample sentence ‘Nowadays I’m feeling totally depressed’ which is related to depression being classified as depression by the model that is deployed in Flask.

The below Figure 4.11 shows the suicide-related sentence ‘I want to kill myself’ being correctly predicted as suicide. The input is given as a text field of a form that uses a flask server that has deployed a model to predict labels.

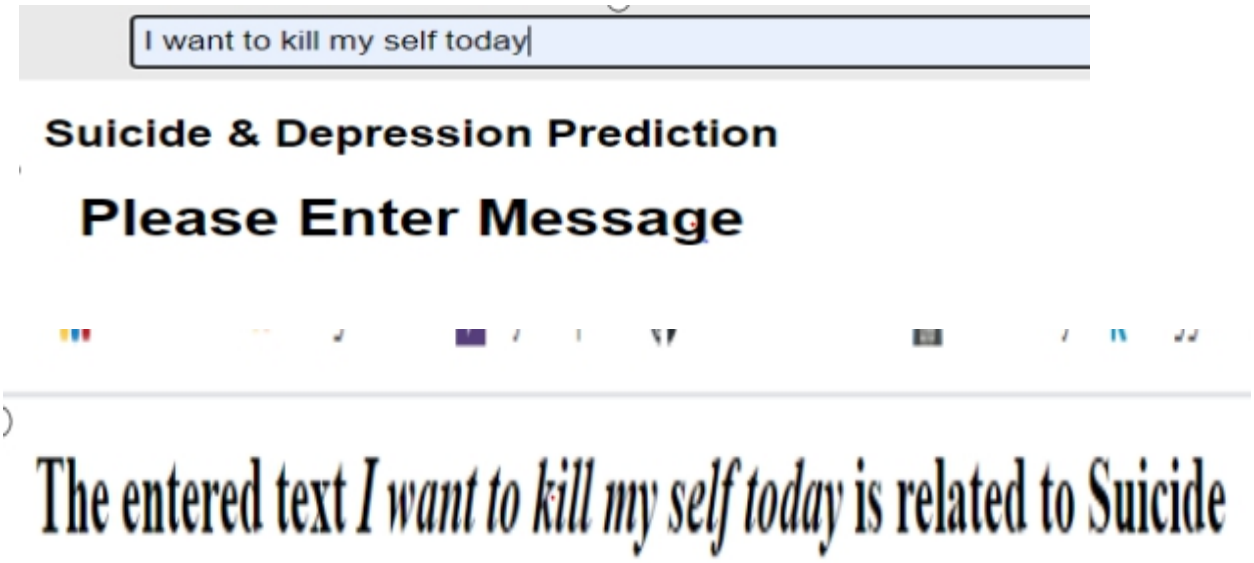


Fig 4.11 Sample Prediction-2

Figure 4.12 shows a normal-conversation (text) 'I like movies' being correctly identified as Normal-Conversation.

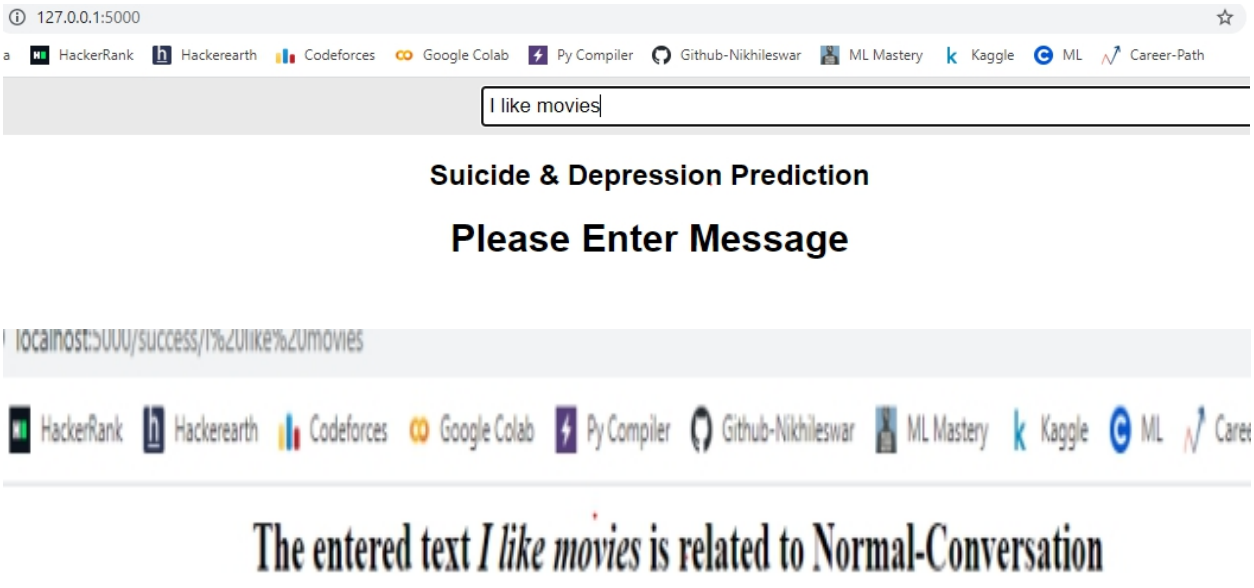


Fig 4.12 Sample Prediction-3

Fig 4.13 Sample Prediction-4

The above Figure 4.13 shows the sentence 'I am going to god' which is related to suicide being incorrectly predicted as Normal-Conversation, this can be due to the training data which may not have texts relating suicide to god.

From Figure 4.14 shown below having the sentence 'Killing is a crime' related to non-suicide the model correctly identifies the sentence as normal-conversation even though the sentence has the word 'kill'. It shows that the model is not over-fitted as most suicide words contained that word.

Fig 4.14 Sample Prediction-5

Fig 4.15 Sample Prediction-6

The model correctly predicts the text shown in Figure 4.15 as a normal-conversation even though the text contains the word ‘depression’ showing that the model is not over-fitting.

4.3 Observations from the Work

For the machine learning algorithms like SVM and XGBoost, those using Universal Sentence Encoder as the encoding technique have produced better results compared to their rivals that are using Term Frequency Inverse Document Frequency + Count-Vectorizer (TF-IDF + CV). Most of the algorithms have produced lower precision values for the 'Suicide' label and lower recall values for the 'Normal-Conversation' label. While the deep learning models like LSTM and LSTM-CNN produced better results with the Word2Vec Word Embeddings compared to the proposed Universal Sentence Encodings, This is due to the fact that LSTM layers require sequential data, Word2Vec encodes text at word level and maintains the sequence of words, While USE encodes text at greater than word-level such encodings are not ideal for the LSTM models. The superior performance of algorithms using USE as an encoder can be accounted for with its optimization to work with greater than sentence texts.

CHAPTER-5

CONCLUSIONS AND FUTURE STUDY

5.1 Conclusion

The project's main motive is to detect suicide-related content on social media forums very effectively to help early suicide detection. This study proposes a better approach that outperforms other baseline models. The high performance of the method can be attributed to the Universal Sentence Encoder, which is being used to encode the text and provide input to the Feed Forward Neural Network. The limitation of the work can be found in the data deficiency. Also, This work assumes that the posts mentioned in the 'SuicideWatch' subreddit are potential suicidal content, and posts from the 'teenagers' subreddit are non-suicidal content that cannot always be true. Since the data is vast, human annotations for the data are not performed. The present study only focuses on the classification of text data written in the English language. In the future, more supported languages can be added. The introduction of manual data annotations can help in improving the results.

5.2 Future Study

- At present only the English language can be classified, in the future, more supporting languages can be added.
- The models work cannot be evaluated on real-time data from Twitter, collection of data from multiple social media platforms and just related to 'Reddit' can be done
- Human labeling of the training data can be very useful as the work assumes that posts from 'SuicideWatch' are related to suicide and posts from the 'teenagers' subreddit are related to normal-conversations which cannot be true always.
- Future studies can be concentrated on creating a more efficient dataset.
- The model works only on data present in the form of text, future work can be extended to identify text contained in images as popular social media platforms like Facebook and Instagram have posts only having images. Hence these also contain suicide-related text present in form of images.

REFERENCES

[1] World Health Organization. National Suicide Prevention Strategies: Progress, Examples, and Indicators; World Health Organization: Geneva, Switzerl, and, 2018.

[2] Beck, Aaron T., Maria Kovacs, and Arlene Weissman. "Hopelessness, and suicidal behaviour: An overview." Jama 234.11 (1975): 1146-1149.

[3] Silver, Michael A., et al. "Relation of depression of attempted suicide, and seriousness of intent." Archives of General Psychiatry 25.6 (1971): 573-576.

[4] Yang, Yang, et al. "Convolutional Neural Networks for Fake News Detection." arXiv preprint arXiv:1806.00749 2.6 (2018).

[5] Mikolov, Tomáš, et al. "Recurrent neural network based language model." Eleventh annual conference of the international speech communication association. 2010.

[6] Mikolov, Tomas, et al. "Distributed representations of words, and phrases, and their compositionality." arXiv preprint arXiv:1310.4546 (2013).

[7] Cer, Daniel, et al. "Universal sentence encoder." arXiv preprint arXiv:1803.11175 (2018).

[8] Tadesse, Michael Mesfin, et al. "Detection of suicide ideation in social media forums using deep learning." Algorithms 13.1 (2020): 7.

[9] Ji, Shaoxiong, et al. "Supervised learning for suicidal ideation detection in online user content." Complexity 2018 (2018).

[10] Vioules, M. Johnson, et al. "Detection of suicide-related posts in Twitter data streams." IBM Journal of Research, and Development 62.1 (2018): 7-1.

[11] Kumar, Mrinal, et al. "Detecting changes in suicide content manifested in social media following celebrity suicides." Proceedings of the 26th ACM conference on Hypertext & Social Media. 2015.

[12] Shuai, Hong-Han, et al. "A comprehensive study on social network mental disorders detection via online social media mining." IEEE Transactions on Knowledge, and Data Engineering 30.7 (2017): 1212-1225.

[13] Huang, Yan, Xiaoqian Liu, and Tingshao Zhu. "Suicidal Ideation Detection via Social Media Analytics." International Conference on Human Centered Computing. Springer, Cham, 2019.

[14] Nikhileswar Komati. (2021, January). Suicide, and Depression Detection, Version 14. Retrieved on May 20,2021,from https://www.kaggle.com/nikhileswarkomati/suicide-watch/version/14.

[15] Bird, Steven, Ewan Klein, and Edward Loper. Natural language processing with Python: analyzing text with the natural language toolkit. " O'Reilly Media, Inc.", 2009.

[16] Mikolov, Tomas, et al. "Distributed representations of words, and phrases, and their compositionality." arXiv preprint arXiv:1310.4546 (2013).

[17] Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." The journal of machine learning research 15.1 (2014): 1929-1958.

[18] Goodfellow, Ian, et al. Deep learning. Vol. 1. No. 2. Cambridge: MIT press, 2016.

[19] Norouzi, Mohammad, Mani Ranjbar, and Greg Mori. "Stacks of convolutional restricted boltzmann machines for shift-invariant feature learning." 2009 IEEE Conference on Computer Vision, and Pattern Recognition. IEEE, 2009.

[20] Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

[21] Brownlee, Jason. Deep learning for time series forecasting: predict the future with MLPs, CNNs, and LSTMs in Python. Machine Learning Mastery, 2018.

[22] Friedman, Jerome H. "Greedy function approximation: a gradient boosting machine." Annals of statistics (2001): 1189-1232.

[23] Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." Proceedings of the 22nd acm sigkdd international conference on knowledge discovery, and data mining. 2016.

[24] Ikonomakis, M., Sotiris Kotsiantis, and V. Tampakas. "Text classification using machine learning techniques." WSEAS transactions on computers 4.8 (2005): 966-974.

[25] Wang, Ziqiang, and Xu Qian. "Text categorization based on LDA, and SVM." 2008 International Conference on Computer Science, and Software Engineering. Vol. 1. IEEE, 2008.

[26] Manning, C. D., P. Raghavan, and H. Schütze. "Xml retrieval." Introduction to Information Retrieval.. Cambridze University Press, 2008.

[27] Joachims, Thorsten. "Text categorization with support vector machines: Learning with many relevant features." European conference on machine learning. Springer, Berlin, Heidelberg, 1998.

[28] De Choudhury, Munmun, Scott Counts, and Eric Horvitz. "Predicting postpartum changes in emotion, and behavior via social media." Proceedings of the SIGCHI conference on human factors in computing systems. 2013.

[29] Beck, Aaron T., Maria Kovacs, and Arlene Weissman. "Hopelessness, and suicidal behavior: An overview." Jama 234.11 (1975): 1146-1149.

[30] American Foundation for Suicide Prevention (AFSP). [Online]. Available: https://afsp.org

[31] Vioules, M. Johnson, et al. "Detection of suicide-related posts in Twitter data streams." *IBM Journal of Research and Development* 62.1 (2018): 7-1.

[32] Desmet, Bart, and VéRonique Hoste. "Emotion detection in suicide notes." *Expert Systems with Applications* 40.16 (2013): 6351-6358.

[33] O'dea, Bridianne, et al. "Detecting suicidality on Twitter." *Internet Interventions* 2.2 (2015): 183-188.

[34] Wang, Chenglong, Feijun Jiang, and Hongxia Yang. "A hybrid framework for text modeling with convolutional RNN." *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 2017.

[35] Wang, Ning, et al. "Learning Models for Suicide Prediction from Social Media Posts." *arXiv preprint arXiv:2105.03315* (2021).

[36] Sawhney, Ramit, et al. "A Time-Aware Transformer Based Model for Suicide Ideation Detection on Social Media." *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2020.

[37] https://github.com/Nikhileswar-Komati/Suicide_Ideation

Appendix

#Data Extraction Using Pushshift API

```
import pandas as pd
import requests #Pushshift accesses Reddit via an url so this is needed
import json #JSON manipulation
import csv #To Convert final table into a csv file to save to your machine
import time
import datetime
 #The Pushshift API can be accessed through building an URL with the relevant parameters wit
 hout even needing Reddit credentials.#These are some examples. You can follow the links and
 they will generate a page with JSON data
 test_url = "https://api.pushshift.io/reddit/search/submission/?&after=1609505059&before=160
 9520400&subreddit=SuicideWatch"
 data = getPushshiftData(1268811603, 1609520400, 'SuicideWatch')
 #Adapted from this https://gist.github.com/dylankilkenny/3dbf6123527260165f8c5c3bc3ee331
 b#This function builds an Pushshift URL, accesses the webpage and stores JSON data in a nest
 ed list
 def getPushshiftData(after, before, sub):
    #Build URL
    url = 'https://api.pushshift.io/reddit/search/submission/?&size=1000&after='+str(after)+'&be
    fore='+str(before)+'&subreddit='+str(sub)
    #Print URL to show user
    print(url)
    #Request URL
    r = requests.get(url)
    #Load JSON data from webpage into data variable
    data = json.loads(r.text)
    #return the data element which contains all the submissions data
    # print("No Error")
```

```
    return data['data']
#This function will be used to extract the key data points from each JSON result
def collectSubData(subm):
    #subData was created at the start to hold all the data which is then added to the global subSta
 ts dictionary.
    subData = list() #list to store data points
    title = subm['title']
    sub_id = subm['id']
    score = subm['score']
    created = datetime.datetime.fromtimestamp(subm['created_utc']) #1520561700.0
    numComms = subm['num_comments']
    selftext = subm['selftext']
    subreddit = subm['subreddit']
    over_18 = subm['over_18']
    #Put all data points into a tuple and append to subData
    subData.append((sub_id,title,selftext,score,created,numComms,over_18,subreddit))
    #Create a dictionary entry of current submission data and store all data related to it
    subStats[sub_id] = subData
#Create your timestamps and queries for your search URL#https://www.unixtimestamp.com/in
dex.php > Use this to create your timestamps
after = "1230768000"
#Submissions after this timestamp (1577836800 = 01 Jan 20)1229385600
before = "1609520400"
#Submissions before this timestamp (1607040000 = 04 Dec 20)#Keyword(s) to look for in sub
missions
sub = "depression"
 #Which Subreddit to search in
#subCount tracks the no. of total submissions that are collected
subCount = 0
#subStats is the dictionary where  the data will be stored.
```

```
subStats = {}
# This function needs to be run outside the loop first to get the updated after variable
data = getPushshiftData(after, before, sub)
# Will run until all posts have been gathered i.e. When the length of data variable = 0# from the
 'after' date up until before date
while len(data) > 0:
#The length of data is the number submissions (data[0], data[1] etc), once it hits zero (after and
 before vars are the same) end
    for submission in data:
        try:
            collectSubData(submission)
            subCount+=1
        except:
            continue
    # Calls getPushshiftData() with the created date of the last submission
    # print(len(data))
    # print(str(datetime.datetime.fromtimestamp(data[-1]['created_utc'])))
    #update after variable to last created date of submission
    after = data[-1]['created_utc']
    # print("CHECK")
    #data has changed due to the new after variable provided by above code
    try:
      data = getPushshiftData(after, before, sub)
      print(subCount)
    except:
      while 1:
        if int(after) >= int(before):
            break
        try:
            print(str(datetime.datetime.fromtimestamp(data[-1]['created_utc'])))
```

```
            after += 10000000
            data = getPushshiftData(after, before, sub)
            print(subCount)
            break
        except:
            after += 10000000
print(len(data))
print(str(len(subStats)) + " submissions have added to list")
print("1st entry is:")
print(list(subStats.values())[0][0][1] + " created: " + str(list(subStats.values())[0][0][5]))
print("Last entry is:")
print(list(subStats.values())[-1][0][1] + " created: " + str(list(subStats.values())[-1][0][5]))

def updateSubs_file():
    upload_count = 0
    #location = "\\Reddit Data\\" >> If you're running this outside of a notebook you'll need this to direct to a specific location
    print("input filename of submission file, please add .csv")
    filename = input() #This asks the user what to name the file
    file = filename
    with open(file, 'w', newline='', encoding='utf-8') as file:
        a = csv.writer(file, delimiter=',')
        headers = ["Post_iD","Title","Body","Score","Publish_date","Total_no_of_comments", "Over_18", "Subreddit"]
        a.writerow(headers)
        for sub in subStats:
            a.writerow(subStats[sub][0])
            upload_count+=1

        print(str(upload_count) + " submissions have been uploaded")updateSubs_file()
```

```
import pandas as pd
import nltk
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection
import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.metrics import accuracy_score, classification_report
from xgboost import XGBClassifier
from nltk.corpus import stopwords
nltk.download('stopwords')
nltk.download('punkt')
from nltk.tokenize import word_tokenize
import os, re
import tensorflow_hub as hub
import tensorflow as tf
embed = hub.load("https://tfhub.dev/google/universal-sentence-encoder/4")
```

```
os.environ['KAGGLE_USERNAME'] = "nikhileswarkomati"
os.environ['KAGGLE_KEY'] = "001b3a30170775e55950edb6ff0c9b17"
!kaggle datasets download -d nikhileswarkomati/suicide-watch
!unzip '/content/suicide-watch.zip'
data = pd.read_csv('/content/SuicideAndDepression_Detection.csv', lineterminator = '\n')data.sa
mple(5)
```

```
def preprocess(string):
    phrase = str(string)
    phrase = re.sub('[^a-z]+', ' ', phrase, flags = re.IGNORECASE)
    phrase = re.sub('(\s+)', ' ', phrase)
    phrase = re.sub('http\S+', ' ', phrase)
    phrase = phrase.lower()
    words_li = ['filler']
    li = list(stopwords.words()) + words_li
    text_tokens = word_tokenize(phrase)
     return " ".join([word for word in text_tokens if word not in li])
print(data['text'].apply(lambda x: len(x.split(' '))).sum())
data['text'] = data['text'].map(lambda string: preprocess(string))
X = data.loc[(data['class'] != 'depression'), 'text'].values
y = data.loc[(data['class'] != 'depression'), 'class'].values
le = LabelEncoder()
y = le.fit_transform(y)
train_X, test_X, train_y, test_y = train_test_split(X, y, train_size = 0.5, stratify = y)
train_X, val_X, train_y, val_y = train_test_split(train_X, train_y, train_size = 0.2, stratify = train
_y)
print(train_X.shape, test_y.shape, val_X.shape)
```

```
embeddings = embed(data.loc[(data['class'] != 'depression'), 'text'][:10000].values)
train_y = le.fit_transform(data.loc[(data['class'] != 'depression'), 'class'][:10000].values)
test_embeddings = embed(data.loc[(data['class'] != 'depression'), 'text'][10000:15000].values)
test_y = le.transform(data.loc[(data['class'] != 'depression'), 'class'][10000:15000].values)
```

#SVM With USE

```
from sklearn.linear_model import SGDClassifier
```

```
sgd = SGDClassifier()
sgd.fit(embeddings, train_y)
print("---------TRAINING_________DONE----------")
y_pred = sgd.predict(test_embeddings)
print(accuracy_score(y_pred, test_y))
print(classification_report(test_y, y_pred))
```

#XGBoost With USE

```
xgb = XGBClassifier()
xgb.fit(embeddings, train_y)
print("---------TRAINING_________DONE----------")
y_pred = xgb.predict(test_embeddings)
print(accuracy_score(y_pred, test_y))
print(classification_report(test_y, y_pred))
```

#XGBoost With TF-IDF + CV

```
xgb1 = Pipeline([('vect', CountVectorizer()),
                 ('tfidf', TfidfTransformer()),
                 ('clf', XGBClassifier()),
                ])
xgb1.fit(train_X, train_y)
y_pred = xgb1.predict(test_X)
print('accuracy %s' % accuracy_score(y_pred, test_y))
print(classification_report(test_y, y_pred))
```

#SVM With TF-IDF + CV

```
from sklearn.linear_model import SGDClassifier
sgd2 = Pipeline([('vect', CountVectorizer()),
                 ('tfidf', TfidfTransformer()),
                 ('clf', SGDClassifier()),
```

```
            ])
sgd2.fit(train_X, train_y)
y_pred = sgd2.predict(test_X)
print('accuracy %s' % accuracy_score(y_pred, test_y))
print(classification_report(test_y, y_pred))
```

#FCNN With USE

```
import itertools
%matplotlib inline
from sklearn.preprocessing import LabelBinarizer, LabelEncoder
from sklearn.metrics import confusion_matrix
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, Embedding
from keras.preprocessing import text, sequence
from keras import utils
checkpoint = tf.keras.callbacks.ModelCheckpoint('model.h5', monitor='val_accuracy', save_bes
t_only=True, verbose=1)
earlystopping = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=5, verbose=
1)
# max_words = 750# tokenize = text.Tokenizer(num_words=max_words, char_level=False)# t
okenize.fit_on_texts(train_X) # only fit on train
# x_train = tokenize.texts_to_matrix(train_X)# x_test = tokenize.texts_to_matrix(test_X)
num_classes = 2
y_train = utils.to_categorical(train_y, num_classes)
y_test = utils.to_categorical(test_y, num_classes)
batch_size = 32
epochs = 10
# Build the model
model = Sequential()
```

```
model.add(Dense(512, input_shape=(512,)))
model.add(Activation('relu'))
model.add(Dropout(0.3))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
              history = model.fit(embeddings, y_train,
              batch_size=batch_size,
              epochs=20,
              verbose=1,
              callbacks = [checkpoint, earlystopping],
              validation_split=0.1)
from keras.models import load_model
model = load_model('/content/model.h5')
y_pred = model.predict(test_embeddings)
y_pred = np.argmax(y_pred, axis = 1)
y_test = np.argmax(y_test, axis = 1)
print(accuracy_score(y_pred, y_test))
print(classification_report(y_pred, y_test))
```

#LSTM With Word Embeddings

```
from keras.layers import LSTM
from keras.preprocessing import sequence
from tensorflow.keras.preprocessing.text import Tokenizer
import itertools
%matplotlib inline
```

```
from sklearn.preprocessing import LabelBinarizer, LabelEncoder
from sklearn.metrics import confusion_matrix
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, Embedding
from keras.preprocessing import text, sequence
from keras import utils
embedding_vecor_length = 32
vocab_size = 10000
max_length = 600
oov_tok = '<OOV>'
trunc_type = 'post'
tokenizer = Tokenizer(num_words = vocab_size, oov_token = oov_tok)
tokenizer.fit_on_texts(train_X)
word_index = tokenizer.word_index
sequences = tokenizer.texts_to_sequences(train_X)
padded = sequence.pad_sequences(sequences, maxlen = max_length, truncating = trunc_type)
#model
model = Sequential()
model.add(Embedding(vocab_size, embedding_vecor_length, input_length = max_length))
model.add(LSTM(100, dropout = 0.3, recurrent_dropout = 0.3))
model.add(Dense(1, activation = 'sigmoid'))
model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
print(model.summary())

checkpoint = tf.keras.callbacks.ModelCheckpoint('model_lstm.h5', monitor='val_accuracy', sav
e_best_only=True, verbose=1)
earlystopping = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=5, verbose=
1)
callbacks_list = [checkpoint, earlystopping]
```

```
model.fit(padded, train_y, epochs=10, batch_size=256,verbose = 1,callbacks = callbacks_list,validation_split = 0.1)

from keras.models import load_model
model = load_model('/content/model_lstm.h5')

testing_sequences = tokenizer.texts_to_sequences(test_X)
testing_padded = sequence.pad_sequences(testing_sequences, maxlen = max_length) y_pred = model.predict(testing_padded)
print(accuracy_score(np.rint(y_pred), test_y))
print(classification_report(np.rint(y_pred), test_y))
#LSTM With USE
from keras.layers import LSTM, Conv1D, MaxPooling1D
X_train = embeddings
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
model = Sequential()
model.add(LSTM(100, dropout = 0.2, recurrent_dropout = 0.2, input_shape = (512, 1)))model.add(Dense(1, activation = 'sigmoid'))
model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
print(model.summary())
checkpoint = tf.keras.callbacks.ModelCheckpoint('model.h5', monitor='val_accuracy', save_best_only=True, verbose=1)
earlystopping = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=5, verbose=1)
callbacks_list = [checkpoint, earlystopping]model.fit(X_train, train_y, epochs=20, batch_size = 32,verbose = 1, callbacks = callbacks_list, validation_split = 0.1)

X_test = test_embeddings
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
from keras.models import load_model
```

```
model = load_model('/content/model.h5')
y_pred = model.predict(X_test)
y_pred = np.rint(y_pred)
y_test = test_y
print(accuracy_score(y_pred, y_test))
print(classification_report(y_pred, y_test))
```

#LSTM-CNN With USE

```
X_train = embeddings
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
model = Sequential()
# model.add(LSTM(100, dropout = 0.2, recurrent_dropout = 0.2, input_shape = (X_train.shape[1], 1)))
# model.add(Dense(1, activation = 'sigmoid'))
model.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu', input_shape = (512, 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(LSTM(100, dropout = 0.1, recurrent_dropout = 0.1))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
print(model.summary())
checkpoint = tf.keras.callbacks.ModelCheckpoint('model_lstm.h5', monitor='val_accuracy', save_best_only=True, verbose=1)
earlystopping = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=5, verbose=1)
callbacks_list = [checkpoint, earlystopping]model.fit(X_train, train_y, epochs=20, batch_size = 32,verbose = 1, callbacks = callbacks_list, validation_split = 0.1)
```

```
X_test = test_embeddings
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

```
from keras.models import load_model
model = load_model('/content/model_lstm.h5')
y_pred = model.predict(X_test)
y_pred = np.rint(y_pred)
y_test = test_y
print(accuracy_score(y_pred, y_test))
print(classification_report(y_pred, y_test))
```

#LSTM with Word Embeddings

```
embedding_vecor_length = 32
vocab_size = 10000
max_length = 600
oov_tok = '<OOV>
'trunc_type = 'post'
tokenizer = Tokenizer(num_words = vocab_size, oov_token = oov_tok)
tokenizer.fit_on_texts(train_X)
word_index = tokenizer.word_index
sequences = tokenizer.texts_to_sequences(train_X)
padded = sequence.pad_sequences(sequences, maxlen = max_length, truncating = trunc_type)

model = Sequential()
model.add(Embedding(vocab_size, embedding_vecor_length, input_length = max_length))mode
l.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(LSTM(100, dropout = 0.3, recurrent_dropout = 0.3))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

```
checkpoint = tf.keras.callbacks.ModelCheckpoint('model_lstm.h5', monitor='val_accuracy', save
_best_only=True, verbose=1)
earlystopping = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=5, verbose=1)
callbacks_list = [checkpoint, earlystopping]
model.fit(padded, train_y, epochs=10, batch_size=256,verbose = 1,callbacks = callbacks_list,vali
dation_split = 0.1)
```

```
from keras.models import load_model
model = load_model('/content/model_lstm.h5')
testing_sequences = tokenizer.texts_to_sequences(test_X)
testing_padded = sequence.pad_sequences(testing_sequences, maxlen = max_length)
y_pred = model.predict(testing_padded)
print(accuracy_score(np.rint(y_pred), test_y))
print(classification_report(np.rint(y_pred), test_y))
```

#FCNN With Word Embeddings

```
checkpoint = tf.keras.callbacks.ModelCheckpoint('model.h5', monitor='val_accuracy', save_bes
t_only=True, verbose=1)
earlystopping = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=5, verbose=
1)

model = Sequential()
model.add(Dense(512, input_shape = (300, )))
model.add(Activation('relu'))model.add(Dropout(0.2))
model.add(Dense(1))model.add(Activation('sigmoid'))# num_classes = 2# y_train = utils.to_c
ategorical(train_y, num_classes)# y_test = utils.to_categorical(test_y, num_classes)
batch_size = 256epochs = 50

model.compile(loss='binary_crossentropy',
              optimizer='adam',
```

```
        metrics=['accuracy'])print(model.summary())history = model.fit(
                        X_train_word_average, train_y,
                batch_size=batch_size,
                epochs=50,
                verbose=1,
                callbacks = [checkpoint, earlystopping],
                validation_split=0.1)
```

#Word Cloud Generation

```
# Python program to generate WordCloud
# importing all necessery modules
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import pandas as pd
# Reads 'Youtube04-Eminem.csv' file
comment_words = '' stopwords = set(STOPWORDS)
# iterate through the csv file
for val in data.loc[(data['class'] == 'SuicideWatch'), 'text'][:200]:
        # typecaste each val to string
        val = str(val)
        # split the value
        tokens = val.split()
        # Converts each token into lowercase
        for i in range(len(tokens)):
                tokens[i] = tokens[i].lower()
        comment_words += " ".join(tokens)+" "
        wordcloud = WordCloud(width = 800, height = 800,
                                background_color ='white',
                                stopwords = stopwords,
                                min_font_size = 10).generate(comment_words)
```

```
# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```

#Deployment Using Flask to extract and classify twitter data

```
from flask import Flask, render_template, request, redirect, url_for
from joblib import load
from get_tweets import get_related_tweets
pipeline = load("suicide.joblib")
def transform(value):
    if value == 0:
        return 'Suicide'
    elif value == 1:
        return 'Depression'
    return 'Normal_Conversation'
def requestResults(name):
    tweets = get_related_tweets(name)
    tweets['prediction'] = pipeline.predict(tweets['tweet'])
    tweets['prediction'] = tweets['prediction'].apply(lambda x: transform(x))
    data = '\n' + str(tweets.prediction.value_counts()) + '\n\n'
    return "The entered Keyword is " + name + data + str(tweets)
app = Flask(__name__)
@app.route('/')
def home():
    return render_template('suicide.html')
@app.route('/', methods=['POST', 'GET'])
def get_data():
```

```
    if request.method == 'POST':
        user = request.form['search']
        return redirect(url_for('success', name=user))
@app.route('/success/<name>')
def success(name):
    return "<xmp>" + str(requestResults(name)) + " </xmp> "
if __name__ == '__main__' :
     app.run(debug=True)
```

#Model Deployment for Single text classification

```
from flask import Flask, render_template, request, redirect, url_for
from joblib import load
from get_tweets import get_related_tweets
pipeline = load("suicide.joblib")
def requestResults(name):
    li = list()
    li.append(name)
    return pipeline.predict(li)
app = Flask(__name__)
@app.route('/')
def home():
    return render_template('index.html')
@app.route('/', methods=['POST', 'GET'])
def get_data():
    if request.method == 'POST':
        user = request.form['search']
        return redirect(url_for('success', name = user))
@app.route('/success/<name>')
def success(name):
    val = requestResults(name)
```

```
    if val == 0:
        text = 'Suicide'
    elif val == 1:
        text = 'Depression'
    else:
        text = 'Normal-Conversation'
    return "<center><h2> The entered text <i>" + name +"</i> is related to " + text + "
</h2></center>"
if __name__ == '__main__' :
    app.run(debug=True)
```

#Extracting Tweets from Twitter

```
import tweepy
import time
import pandas as pd
# api key
api_key = "YfsfRUtv0Jstlvm0TLg8DrZNA"
# api secret key
api_secret_key = "Dx95SabGPVACrlQanwkajOnsfss0tWsyej8xO8rUKnf6N70Tyh"
# access token
access_token = "704330902432669696-pmTtYoAM3ywia3zAY5sWAEVkzhWUwan"
# access token secret
access_token_secret = "BSW1LmSmDZmNrDPL3KytWXgZeOTHo99Ee1vDu1FBc5EAJ"
authentication = tweepy.OAuthHandler(api_key, api_secret_key)
authentication.set_access_token(access_token, access_token_secret)
api = tweepy.API(authentication, wait_on_rate_limit=True)
def get_related_tweets(text_query):
    # list to store tweets
    tweets_list = []
    # no of tweets
```

```
    count = 100
    try:
        # Pulling individual tweets from query
        for tweet in api.search(q=text_query, count=count, lang = 'en'):
            print(tweet.text)
            # Adding to list that contains all tweets
            tweets_list.append({'created_at': tweet.created_at,
                                'tweet': tweet.text})
        return pd.DataFrame.from_dict(tweets_list)
    except BaseException as e:
        print('failed on_status,', str(e))
        time.sleep(3)
```