Statistical methods to predict future risk of suicidal ideation from social media data

by

Tyler Bastian

B.S., Kansas State University, 2019

A REPORT

submitted in partial fulfillment of the
requirements for the degree

Master of Science

Department of Statistics
College of Arts and Sciences

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2022

Approved by:

Major Professor
Dr. Perla E. Reyes Cuellar

Abstract

Suicide, the act of taking ones own life, is a tragedy for all involved and a public health concern in the United States. Suicide is the tenth leading cause of death in the United States which makes the monitoring of suicide and suicidal ideation, or the process of thinking or ruminating ones own death, crucial in the interest of public health. With the rapid development of machine learning methods, new analyses of social media data to predict individual suicide risk and behavior have been reported. A recent study, "A Machine Learning Approach Predicts Future Risk to Suicidal Ideation From Social Media" by Roy et al. (2020) https://doi.org/10.1038/s41746-020-0287-6, showed promising results in classifying Twitter users into a suicidal category 4, 7, 14, and 21 days in advance of expressing suicidal ideation.

Roy et al. propose training a set of neural networks to detect and score psychological constructs associated with suicidal thoughts. Using the obtained scores as inputs, they implement a Random Forest algorithm to determine individual Twitter users risk of future suicidal ideation. In this report, we offer a detailed explanation of methodology used by Roy et al. alongside evaluating the reproducibility of their work. We first extract data from Twitter and train a series of neural networks to identify if a tweet expresses psychological constructs associated with suicidal thoughts which include; burden, stress, anxiety, loneliness, insomnia, hopelessness, and depression. Using 1.2 million tweets from N = 182 suicidal ideation (SI) cases and 30,648 tweets from 347 controls, we then train a Random Forest model using neural network outputs to predict a binary outcome of SI status. The model predicted within 7 days N = 78 SI events derived from an independent set of 342 suicidal ideators relative to N = 3,458 non-SI tweets with an AUC of 0.83, slightly lower than Roy et al.'s 7 day model prediction having an AUC of 0.88. Algorithmic approaches such as this could be applied to potentially identify an individuals future risk of suicidal ideation and could be integrated into medical technologies to aid in suicide screening and risk monitoring.

Copyright

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would first like to express my deepest gratitude to Dr. Perla Reyes. Six years ago she showed me great mentorship and professional training through my undergraduate career and agreed to further my professional and educational development throughout my Graduate experience. Working under Dr. Reyes' supervision I had the privilege to benefit from her knowledge, guidance, and most important her passion for statistics and data science. Without her guidance this report would not have been possible.

I would also like to thank my council members who also provided excellent guidance throughout my Graduate program. To Dr. Haiyan Wang, without her time and technical knowledge in data mining and machine learning, my understanding and application of the machine learning techniques used within this report would not be possible. To Dr. Goh, without him my understanding of Bayesian statistics and optimism within the realm of Statistics would not be nearly as developed as it is today. To Dr. Michael Higgins a great thank you is necessary for my acceptance into this program and whose knowledge in sampling methods and bias has helped me in my ethical understanding of the application of statistical techniques.

I would also like to thank my parents who have supported me throughout my undergraduate and graduate training and offered me valuable words of encouragement during trying times.

Dedication

To all my friends who have passed by suicide. You may be gone but you will never be forgotten. May this report serve in the advancement of suicide risk detection and intervention.

Chapter 1

Introduction

1.1 Motivation

In the last couple of decades social media has advanced the expression of ideas, emotions, and thoughts publicly. With the availability of online personal journal entries, blog posts, and comments we have seen an increase in research regarding the automatic detection of emotions and sentiments from textual data.

While most literature available focuses on the detection and assessment of emotions in online textual data, there is less research available regarding the investigation of emotion detection in textual conversation. One can argue that with a large amount of dialogue available online, the detection of emotions hosts new challenges when compared to the detection of emotions within monologues as different influences affect the dialogue between individuals. These different influences can range from the input of opinion from another person outside the dialogue to personal events or trauma that may occur. In this report, we investigate the effectiveness of neural networks for the task of emotion detection and classification in short dialogues with the goal of identifying warning signs of emergency that could be used to trigger timely preventative interventions.

It comes to no surprise that automatically monitoring platforms for mental health screening is beneficial. With new research being done in the monitoring of mental health among

social media users, there still remains a gap in research where the time elapsed between the first signs of mental health issues, and the detection and intervention for a victim plays a crucial role. Therefore, the earlier a possible harmful event is detected the better. However, in recent years, besides a machine learning program developed by Roy et al. (2020), not much research has been provided to detect mental health issues in a timely manner to reduce the risk of great harm.

With the increasing usage of textual data to provide insight for marketing and business purposes (Jacobs et al., 2018; Van der Aa et al., 2018), stock market predictions (Abdullah et al., 2013), and the movement of the Coronavirus pandemic (Ebadi et al., 2021), it is becoming easier to extract public opinion on services or products that help build relations between business or governments and the public which inevitably leads to more profit.

Techniques to provide sentiment analysis can range from a binary classification of text into positive or negative, to more refined classifications of emotions such as *happy*, *sad*, *angry*, or *excited* (Hirschberg and Manning, 2015). Emotion classification is especially useful within business and marketing settings and a variety of Natural Language Processing (NLP) applications such as, speech-to-text and human-computer interactions that take into account the emotional state of users to provide more human-like responses as it gives companies a chance to monitor market research and competition (Poria et al., 2019).

With the growing number of usages of textual data, we soon may be able to provide better awareness to a variety of emergencies such as the detection of toxicity, hate speech, and cyber-bullying in online platforms (Roy et al., 2020). With the increased awareness to these emergencies, we may be able to provide timely interventions within possible violent situations (Ballesteros et al., 2020).

Within healthcare, online textual data can provide a myriad of uses such as the online detection of a disease outbreak (Chapman et al., 2004), finding drug use patterns (Carrell et al., 2015), and the identification of adverse drug side-effects (Leaman et al., 2010). Another growing usage within healthcare is the automatic detection of mental health issues (Stewart and Velupillai, 2021), a relatively new field that has recently gained attention within Twitter, Facebook, and Reddit. Within these large corporations the application of NLP helps provide

platforms the ability to detect various mental health issues such as anxiety, depression, eating disorders, and suicidal ideation (Le Glaz et al., 2021).

1.2 Objective

The focus of this report is to understand the statistical learning techniques used to detect suicidal ideation in Twitter as proposed by Roy et al. (2020). We believe that a more lengthy and detailed explanation should facilitate its application by others and enhance research around it and its components to reach sooner the goal of creating a diagnostic tool that could be used to trigger interventions and prevention. The method has three main components: the extraction of textual data from the Twitter database, the classification of emotions, and the early detection of suicidal ideation. Since this study will use observations based upon people and their personal data, the International Review Board had approved the usage of personal data within this study. Upon the International Review Board's approval, we use Python version 3.8.5 to conduct our analysis. Twitter users must agree to Twitter's Terms and Services making their data public. However, due to the sensitive aspect of our study, we will be changing all forms of user identification to random identification numbers as to prevent any possibility of identifying individuals within this study. Thus, respecting the privacy of the subjects.

Deep learning approaches have significantly improved what was formerly known as computationally impossible tasks. However, in order to achieve such results it is often relied on large annotated datasets which are in many cases difficult to obtain and rarely capture the true randomness of events. This report explores the use of data extraction to perform such analyses.

For the purpose of model generalizability, the use of handcrafted features is minimized. On the other hand, neural network architectures are an essential part of our experiment as they can automatically extract features. We investigated the effectiveness of our model by experimenting with three different word embeddings: FastText, Glove, and Word2Vec. We applied these three types of pre-trained word embeddings within a Feed Forward Neural

Network (FFNN), Convolutional Neural Network (CNN), and a Long Short-Term Memory Recurrent Neural Network (LSTM). Pre-trained word embeddings are the embeddings that are trained on large datasets, saved, and then used for solving another similar task.

This report aims to investigate the use of these developments of deep learning and benefit from the strength of these methods by applying them to a case of natural language processing in emotion detection and clinical psychology. As automatic assessment tools can provide helpful complimentary measures to monitor emotional state and mental health of online users, as this field gains more traction this report can serve as a starting point for future research in this area.

1.3 Mental Health Assessment

The use of text collected from Twitter, Facebook, Reddit, blogs and online forums has been used by countless researchers as resources for experimentation with classification tasks pertaining to mental health issues.

Pestian et al. (2010) experimented with different machine learning methods for suicide note classification. They focus on developing methods of natural language processing that distinguish between genuine and elicited suicide notes. They hypothesize that machine learning techniques, such as a logistic decision trees, are able to classify suicide notes as well as mental health professionals (Pestian et al., 2010). Using emotions, parts of speech tags, readability, and words as features they achieve a 78% accuracy with with their logistic decision trees.

Shen and Rudzicz (2017) used different feature sets including Word2Vec embedding, Latent Dirichlet allocation topic modelling, lexicon-syntactic features, and N-grams (unigrams and bigrams) to detect anxiety in Reddit posts (Shen and Rudzicz, 2017). Originally, the authors compared the results achieved by a support vector machine and a 2-layer neural network. Though both classifiers performed well, the support vector machine yielded slightly better results. However, they achieved their best result of 98% accuracy using the neural network with n-gram probabilities and word embeddings combined with Linguistic Inquiry

and Word Count (LIWC) features.

More recently, Wshah et al. (2019) used a virtual agent to study the symptoms of psychological distress in dialogues. They show that with the use of an ensemble of support vector machines, Naive Bayes classifiers, logistic regression, and random forest algorithms they can predict elevated Post Traumatic Stress Disorder symptoms receiving an area under the curve (AUC) score of 0.85 . Furthermore, they show that 7-self reported features are needed to obtain this AUC score and provide accurate predictions that can be made 10 to 20 days post trauma.

All previous work used a classic classification approach that does not measure how early detection is performed. Early detection can be used in many applications where intervention is needed and the timing of the intervention is important. An example of such an application is suicidal ideation (Roy et al., 2020). According to Roy et al. (2020), risk assessment approaches are currently focused at detection after the event has occurred, though the timing of the detection can be of unmitigated importance. Therefore, it is important to try to minimize the time between seeing the first sign of a harmful behavior and signaling a warning. Roy et al. (2020) propose using a combination of 9 trained neural networks to classify individual tweets into 9 emotions or psychological constructs determined with the help of professional psychiatrists.

1.4 Methodology

To understand the methods used within this report, once the International Review Board approved the study, we downloaded tweets from Twitter using queries, or specific requests for data from a database, associated with suicidal ideation provided by Roy et al. (2020) We then applied commonly used word embeddings and a series of neural networks to detect psychological constructs associated with suicidal ideation as presented by Roy et al. (2020). These psychological constructs include anxiety, burdensomeness, depression, hopelessness, insomnia, loneliness, and stress. We also include the variables polarity and subjectivity provided by the TextBlob package as extra variables to detect the positive or negative sentiment and

a quantitative subjectivity for each tweet, respectively. Finally, we trained Random Forests to predict future risk of suicidal ideation 4, 7, 14, and 21 days in advance, using the obtained matrix of psychological constructs, polarity, and subjectivity as input.

1.5 Classification Analysis

Here, we present a review of methods related with this work starting with the more general concept of classification to then going into the details of the used techniques in subsequent chapters. The goal of text classification, to assign a piece of text to one or multiple classes, is an essential part of many Natural Language Processing applications. Here we focus on textual classification for emotion detection and for mental health assessment.

The goal of this report is to aid on the development of a tool that can correctly identify users with suicidal ideation (SI) before the event occurs. The methodology proposed by Roy et al. (2020) revolves around the classification of Twitter users into an SI class, days or weeks before they sent an SI tweet. We found it a compelling point that deserved a more detailed explanation. Classification is a data analysis task within data-mining, that identifies and assigns categories to a collection of data to allow for more accurate analyses where the classification method makes use of mathematical techniques such as decision trees, linear programming, neural networks and statistics (James et al., 2013). Classification analysis can be used to question, make a decision, or predict behavior through the use of an algorithm. It works by using a set of training data which contains a certain set of attributes as well as the likely outcome. The job of the classification algorithm is to discover how that set of attributes reaches its conclusion. Such classification analyses involve two steps. The first step involves a learning step or training phase where different algorithms are used to build a classifier by making the model learn by the use of an available training set which provides accurate prediction results. The second step involves a classification step, or testing phase, where the model, used to predict class labels, tests the constructed model on test data which in turn estimates the accuracy of our model (James et al., 2013).

Over the years, a variety of techniques have been used for text classification systems.

Classical machine learning algorithms such as Naive Bayes Classifiers (Xu et al., 2017), Logistic Regression (Pranckevičius and Marcinkevičius, 2016), K-Nearest Neighbors (Yong et al., 2009), Support Vector Machines (Colas and Brazdil, 2006), Decision Trees (Su and Zhang, 2006), and Random Forests (Shah et al., 2020) have been commonly used in the literature for text classification. Although these methods can extract classification rules automatically, they still require predetermined linguistic features as their input (Shah et al., 2020). With the availability of relatively large corpora of annotated data, and thanks to recent advances in the field of deep learning for natural language processing, neural network architectures have gained much interest for the task of test classification (Goldberg, 2017).

1.6 Evaluation Metrics

One essential part of developing a classification system is evaluating how it performs when exposed to new samples. Thus, appropriate evaluation metrics that provide a reliable measure of the system's performance are needed. In this section, a brief summary of the most common evaluation metrics used for classification is presented. Most evaluation metrics are based on a confusion matrix that gives information about the system's prediction for the samples and also the true label for the samples in question which we can use to calculate the models accuracy, precision, recall, and F-measure.

1.6.1 Accuracy

The accuracy measure takes into account the number of samples correctly classified out of all predictions made by the system, irrespective of the samples' class. It is calculated using Equation 1.1.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1.1}$$

In the case of a classification system aimed at diagnostics, accuracy is not a reliable measure, since there is often the necessity to give more weight to positive labels (i.e. at-risk

individuals). As a result, more informative measures such as precision, recall, and F-measure are preferred.

1.6.2 Precision

Precision measures the ratio of correctly classified true or positive samples out of all samples classified as positive and is calculated using Equation 1.2.

$$precision = \frac{TP}{TP + FP} \tag{1.2}$$

1.6.3 Recall

The recall measure, otherwise known as sensitivity, is used to calculate the ratio of correctly detected positive samples out of all positive samples, both detected and undetected by the system in the data set. Equation 1.3 is used to measure recall.

$$recall = \frac{TP}{TP + FN} \tag{1.3}$$

In health applications, recall is a very important metric that measures the capacity of the system to emit alerts when needed and reduce the number of false negatives.

1.6.4 F-Measure

There is a inverse relationship between precision and recall though. When one decreases another increases. A system that has a tendency to predict more negative labels can have a good precision but a low recall, while a system that outputs too many positive labels has a good recall, but low precision.

The traditional F-measure, the F_1 score, is the harmonic mean of precision and recall. It was developed in an attempt to combine both recall and precision into a single measure making it possible for model optimization and selection based on a single evaluation metric.

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{1.4}$$

1.6.5 Receiving Operating Characteristics

A Receiving Operating Characteristics (ROC) curve is a graphical display showing the performance of a classification model at all classification thresholds. The ROC curve uses the true positive rate, or sensitivity rate, on the y-axis and false positive rate, or specificity, on the x-axis. The false positive rate, or specificity can be calculated using Equation 1.5 below.

$$Specificity = \frac{TN}{TN + FP} \tag{1.5}$$

The ROC curve summarizes the trade-off between the true positive rate and false positive rate for a predictive model using different probability thresholds. The true positive rate, also known as recall or sensitivity is calculated using Equation 1.3.

1.6.6 Area Under the Curve

The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve. This area under the ROC curve is always represented by a value between 0 and 1. Because we try to maximize the this area, the higher the AUC the better the system is at distinguishing between positive and negative classes.

1.7 Report Overview

In this chapter we introduced emotion and mental health assessment in textual data, our motivation and methodology to apply natural language processing techniques to address the aforementioned tasks, and presented an introduction to classification techniques and common evaluation metrics. Chapter 2 provides detail over the connection and extraction

of large amounts of textual data from Twitter's database. Chapter 3 provides an account of our initial classification system using feed-forward neural networks, long short-term memory recurrent neural networks, and convolutional neural networks which are developed for the task of contextual emotion detection. Chapter 4 presents the development and evaluation of a more refined ensemble approach aimed to study the usage of a Random Forest algorithm for early detection of suicidal ideation as proposed by Roy et al. (2020). Finally, Chapter 5 concludes this report with a discussion and summary of the performance of the techniques used and an overview in possible future research in this area.

Chapter 2

Tweet Extraction and Text Pre-Processing

2.1 Suicidal Ideation Tweet Extraction

The first step to reproduce the Roy et al. (2020) study is the extraction of textual data from Twitter's Application Programming Interface version 2 (API V2.0) through their provided endpoints. At the end of 2020 Twitter introduced this new API to include more features, extend the data you can pull and analyze, and introduced new endpoints, which are points where the API connects with the program or script. With the introduction of this new API, Twitter also introduced a new powerful free product for academics: The Academic Research product track. This track grants free access to full-archive search and other API V2.0 endpoints with a volume cap of 10,000,000 tweets per month. Though, since API V2.0 is fairly new, fewer resources exist if issues are found through the process of collecting data for research.

In order to send your first request to the Twitter API, a developer account is needed. This developer account grants access to Twitter's API and its various endpoints for data extraction. Once the developer account has been set up, we may begin to access its various endpoints for data.

Search All	https://api.twitter.com/2/tweets/search/all
User Timeline	https://api.twitter.com/2/users/:id/tweets

Table 2.1: *Twitter Academic Research endpoints*

To begin extraction, a "bearer token" is supplied from Twitter to begin work in the application you created upon successfully establishing a developers account. This token allows developers to have a more secure point of entry for using the Twitter APIs and is one of the core features of API V2.0. To simplify, a bearer token is a byte array of an unspecified format that you generate using a script like a curl command or within your account and application information. Storage of this token is highly recommended within an environment variable, such that use of the variable may be used by calling the variable name within any script.

Next, we create headers that will take our bearer token, pass it for authorization and return headers that will be used to access the data needed for analysis. Once we have access to the API, we search individual endpoints, or URLs to extract the data of interest. Table 3.1 supplies the two URLs used in this report to access Twitter and extract information. The full-archive, or search all, endpoint provides access to all tweets dating back to the first tweet in March 2006 which is only available for those approved for the Academic Research access. With this endpoint, there is one required string parameter, "query" which can hold up to 1,024 characters and returns tweets matching the search query. The queries used for data extraction can be found in Table 3.2 Because our interest is for month by month data, we also include the parameters, "start_time" and "end_time" which are in the format 'YYYY-MM-DDTHH:mm:ssZ (ISO 8601/RFC 3339). These parameters provide the newest, most recent UTC timestamp in second granularity and is exclusive. If the end time parameter is the only parameter given, then tweets from 30 days before the specified end time will be returned by default. For this report we specify a start time and end time ranging month to month from 01/01/2019 to 08/31/2021.

We can also request parameters associated with each tweet, which is the data we wish to extract. To utilize this, the Twitter "field" parameter is used which is able to return

the individual user's identification number, the tweet text, the tweet identification number, favorite count, retweet count, in reply to, geographic location or georeference returned in latitude and longitude coordinates, the conversation identification number, the date the tweet was created or posted, the language of the individual user, and its source. Within this report we are specifically interested in the user identification, the date and time of the tweet, the text of the tweet, the source of the tweet (i.e. Twitter for iPhone, Twitter for Android), and the geographic location. Due to Tobler's First Law of Geography which states that objects or occurrences that occur close together are more related than those far apart, one may be interested in grouping by geographic location. Though due to the location settings being denied access within individuals users account, only a handful of geographic locations were returned, therefore we are not able to use this parameter within the scope of this analysis.

Finally, we are able to combine all aforementioned functions to connect to the endpoint. This provides the direct connection and extraction of data and allows for a 'next token' parameter which gives a unique identifier to the next page of results to continue data extraction. Once the data has been successfully returned in a JavaScript Object Notation (JSON) format, we receive a list of dictionaries where each dictionary represents the data for each tweet. Because our results are returned in JSON format, we then convert this format to a Comma-Seperated Values (CSV) format.

2.2 Timeline Extraction

With the functions we have defined previously for suicidal ideation tweet extraction, we are able to loop through results to extract data. After suicidal ideation tweets have been retrieved and converted to a CSV format, we then loop through each individual user's identification number and extract their timelines back to January 01, 2019 where individual users Twitter timelines are returned and converted to CSV format. Since we are interested in tweets up to the tweet where suicidal ideation occurs, each tweets date that was identified using the query, 'I suicide thinking OR planning' was saved and converted to the required format for timeline extraction.

Flag SI Users	"I suicide thinking OR planning"
Control	"I"
Burden	"I am a burden"
Loneliness	"I am lonely"
Stress	"I am stressed OR stress"
Depression	"I am depressed"
Anxiety	"I am anxious or I am nervous"
Hoplessness	"I hopeless"
Insomnia	"I am not sleeping OR I can't sleep OR insomnia"

Table 2.2: *Twitter Queries for Data Extraction*

Next, we change the endpoint of interest to extract timelines based on each individual user id. To extract each user's timeline, a list of user identification numbers was supplied to the endpoint by looping through each users id and attaching it the endpoints URL by formatting the endpoint string to include the i^{th} user id. Once we have formatted the endpoint, we are then able to extract each individual user's timeline that was queried using the query, "I suicide thinking OR planning". We then change all user identification numbers as to follow the International Review Boards guidelines when using real-word data obtained from individuals flagged with suicidal ideation.

Once we have extracted our suicidal ideation tweets and their corresponding user timelines, training tweets must be extracted in order to train a set of neural networks. To do this, we simply switch our endpoint back to a full-archive search and using the same range of dates mentioned previously, extract data using the queries found in Table 2.2. The specific use of queries and training of the emotion classification using neural networks will be further discussed in Chapter **??**.

2.3 Word Embeddings Application

Now that we have collected our Twitter data we must next pre-process the data. Within every tweet collected, we remove all mentions, emoticons, non-English characters, and URL's. Removal of capitalization and punctuation is needed, though stop words are not removed as is traditionally done in sentiment analysis, due to the limited number of characters allowed

in a tweet and the interest in understanding the context of each tweet and classifying it using neural networks.

Traditionally, when analyzing or modeling words, they are represented by one-hot vectors (i.e. a vector of all zeroes, except for an element having the value 1), which indicate the index of a word in a vocabulary. Embeddings are considered as matrices that are used to map these one-hot representations to a dense space with the aim of capturing semantic information about the word and also reducing sparcity (Goldberg, 2017).

The concept of word embeddings was introduced first by Bengio et al. (2003) who trained word embeddings with the neural model itself in order to perform the task of language modeling which is predicting a word given its previous words (Bengio et al., 2003).

Figure 2.1 shows the architecture of the neural word embedder of Bengio et al. (2003), where C is considered as the embedding matrix.

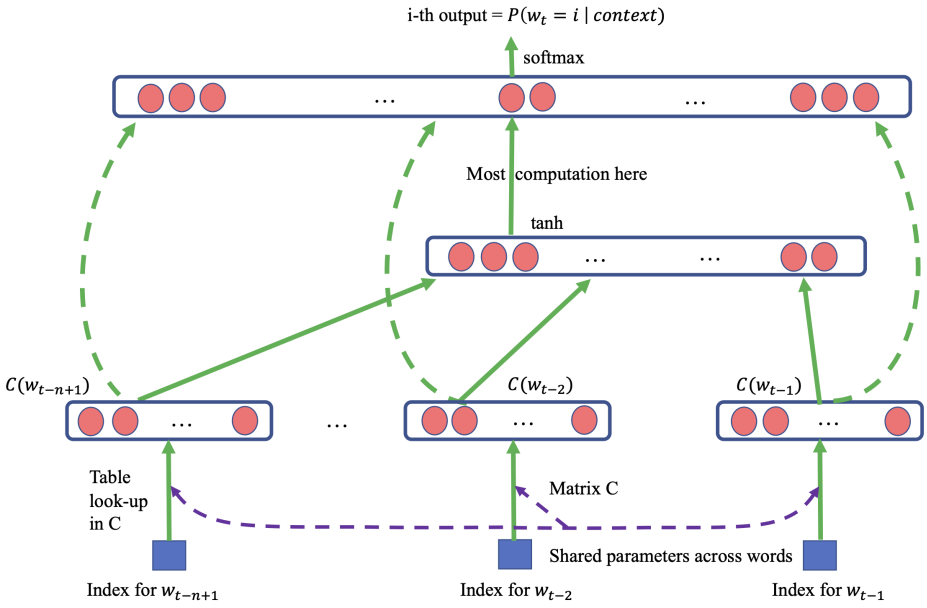

Figure 2.1: *Classic neural language model*

Throughout the years, different methods for training word embeddings have been developed. One of the most popular methods, known as the Word2Vec model, was introduced by Mikolov et al. (2017). The Word2Vec model uses one of the following two approaches for training the embeddings: Continuous Bag of Words (CBOW) and Skip-Gram. In the CBOW

model, the embedding is created by using the words surrounding a target word as input and the target word itself as the output. However, in the Skip-Gram model an opposite approach is used to create and extract an embedding vector with the target word being fed as input, and the surrounding as output. Both CBOW and Skip-gram architectures have an input, projection, and output layers. After the training is complete, the weights that connect the input layer to the projection layer will be considered as the embedding matrix.

GloVe is another method for training word embeddings, developed by Pennington et al. (2014). GLoVe uses statistics of word co-occurences making semantic relationships between words more explicit, unlike Word2Vec which does not as it only considers instances of word co-occurrences (Pennington et al., 2014).

A more recent word embedding, FastText, was developed by Joulin et al. (2016a) and follows a similar idea as Word2Vec but instead of using words to build word embeddings, FastText goes one step further (Joulin et al., 2016a). This deeper level consists of parts of words and characters that provide context, therefore the building blocks of the embedding are individual characters instead of words. There are two major advantages to this approach. First generalization is possible as long as new words have the same characters as known ones. Second, less training data is needed since much more information can be extracted from each piece of text. That is why there are pre-trained FastText models for languages besides English.

In this report, we evaluated the accuracy and precision of a feed-forward neural network, a long short term memory recurrent neural network, and a convolutional neural network with the three aforementioned word embeddings to identify tweets that express specific psychological aspects listed in Section 2.2. We applied each of them to the processed text. Word2Vec pre-trained word embeddings were obtained from Google and were trained on part of a Google News dataset containing about 100 billion words and contains 300 dimensional vectors for 3 million words and phrases. Glove pre-trained word embeddings were obtained from Stanford University's Pennington et al. and contains 100 dimensional vectors for 6 billion tokens obtained using Wikipedia and the fifth edition of Gigaword, a comprehensive archive of newswire text data acquired over several years by the Linguistic Data Consortiume

(Pennington et al., 2014). The glove model was trained on the entires of non-zero of a global word-to-word co-occurrence matrix which arranges how frequently words co-occur with another word in a collection of documents. FastText pre-trained word-embeddings for text classification were obtained from Meta's Facebook Joulin et al. (2016b) and provides 1 million 300 dimensional word vectors trained on Wikipedia 2017, UMBC webbase corpus, and statmt.org news dataset containing 16 billion tokens.

Chapter 3

Emotion Classification

Textual emotion detection has typically been addressed as a multi-class classification task, where a text is classified into different psychological constructs or emotional categories ranging from basic emotions to fine-grained emotional classes. Studies focusing on emotion detection have made use of different corpora and different evaluation metrics.

Dini and Bittar (2016) broke down the task of emotion detection from tweets into a stream of decisions: classifying tweets into emotional and non-emotional categories, then tagging the emotional tweets with the appropriate emotional label (Dini and Bittar, 2016). For the latter, they compared a symbolic system using gazetteers, a geographic index dictionary, regular expressions, and graph transformations with a machine learning system using a linear classifier with words, lemmas, noun phrases, and dependencies as features. Using their collected corpus of emotional tweets, the rule-based approach achieved an F1 score of 0.41 while the machine learning approach yielded an F1 score of 0.58 on 6 emotion classes.

Madisetty and Desarkar (2017) made use of a Support Vector Regression model to determine the intensity of 4 emotions: anger, fear, joy, and sadness in a dataset of tweets that they had previously collected and annotated. As features, they used word and character n-grams, word embeddings using the Word2Vec skip-gram model, and affect-related lexical features. Using the Pearson correlation coefficient as the evaluation metric, they demonstrated that word embeddings yield better results than n-gram features. They achieved their best average

result of 0.66, using a combination of word embeddings and lexical features.

Al-Khatib and El-Beltagy (2017) proposed an approach to detect the intensity of affect in tweets with features that were directly derived from Twitter and included feature vectors extracted using the AffectiveTweets package of the Weka workbench (Al-Khatib and El-Beltagy, 2017). They developed 3 models using different subsets of the feature set as input to either a Support Vector Machine, Naive Bayes classifier, or a Complement Naive Bayes Classifier. Using the manually extracted dataset, they achieved their best score using a Complement Naive Bayes classifier with an accuracy of 68.12% and an F1 score of 0.658.

Khanpour and Caragea (2018) focused on domain-specific emotion detection (Khanpour and Caragea, 2018). They created a dataset of 2107 sentences taken from online forums in the Cancer Survivors Network website and in order to combine the strengths of lexicon-based and machine learning approaches, they proposed a model that uses Word2Vec embeddings as input to a CNN. The CNN generates feature vectors which are then augmented with domain-specific lexical features. The combined features are then used as input to an LSTM network which classifies the text into 6 different emotion categories.

3.1 Deep Learning Approaches

Deep learning algorithms have gained much popularity within the last 10 years due to their success in improving the state-of-the-art in many fields including NLP (Socher et al., 2012). They have shown the capability of modeling complex patterns in large data sets by use of backpropogation which allows the system to automatically find rules and features needed for classification. A neuron in the brain is a cell whose principal function is the collection, processing, and diffusion of electrical signals whose information-processing emerges primarily from the networks of such neurons. Because of this, some early work in artificial intelligence is dedicated to creating artificial neural networks (Russell and Norvig, 2002). Figure 3.1 below shows a simple mathematical model of the neuron devised by McCulloch and Pitts. In general, it "fires" when a linear combination of its inputs exceeds some threshold (McCulloch and Pitts, 1943). Since 1943, much more detailed and realistic models have been developed

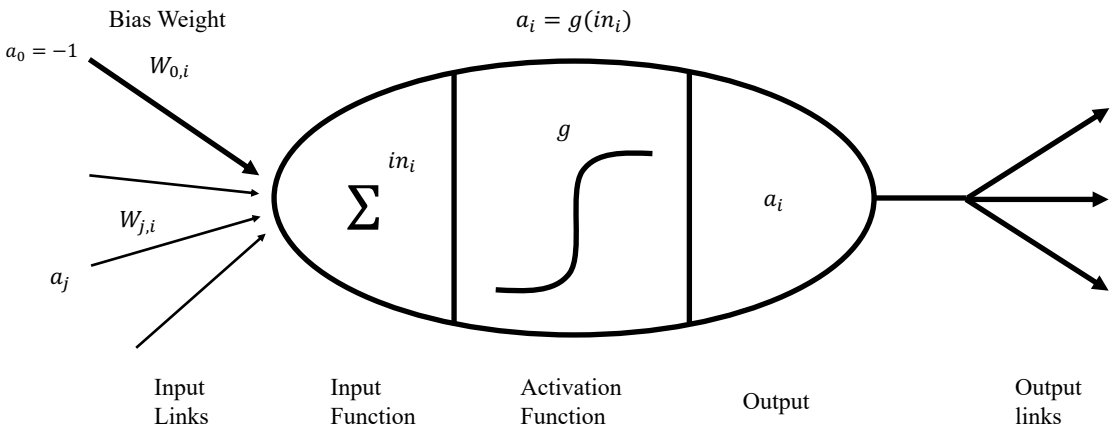

Figure 3.1: *A simple mathematical model for a Neuron. The unit's output activation a_i, where a_j is the output activation of unit j and $W_{j,i}$ is the weight on the link from unit j to this unit.*

for neurons and larger systems in the brain leading to the modern field of computational neuroscience. In other professions, including statistics and artificial intelligence, interest in more properties of neural networks, such as their ability to perform distributed computation, to tolerate noisy input data, and to learn are preferred (Russell and Norvig, 2002).

3.1.1 Units in Neural Networks

Neural networks are composed of nodes or units connected by directed links. One can simply think of a node as something that holds a number with a link from unit j to unit i that serves to propagate the activation a_j from j to i. Each link also has a numeric weight $W_{j,i}$ associated with it, which determines the strength and sign of the connection. (Krogh, 2008) Each unit i first computes a weights sum of inputs:

$$in_i = \sum_{j=0}^{n} W_{j,i} a_j$$

Then it applies an activation function g to this sum to derive the output:

$$a_i = g(in_i) = g(\sum_{j=0}^{n} W_{j,i} a_j)$$

The activation function g is designed to meet two desired effects. First, we want the unit to be "active" (near +1) when the "right" inputs are given, and "inactive" (near 0) when the "wrong" inputs are given. Second the activation needs to be non-linear, otherwise the entire neural network collapses into a simple linear function (Krogh, 2008). Three choices for activation functions, are the sigmoid function (also known as the logistic function seen in Equation 3.1), the tanh function, and the Rectified Linear Unit (ReLu) activation function (Equation 3.2). The sigmoid function has the advantage of being differentiable, which will have a threshold at zero. The tanh function has the advantage of back-propagation which the sigmoid function fails to meet. The sigmoid function has a range from 0 to 1 which allows for the prediction of probability whereas the tanh function ranges from -1 to 1 which allows for a zero centered output and thereby aiding the back-propagation process, though similar to the sigmoid function, fails due to the vanishing gradient problem. The ReLu activation function has rapidly become the default activation function for many neural networks for its computational simplicity and constant derivative, its ability to output a true zero value, and its linear behavior outputting values between 0 and infinity, all of which help it to overcome the vanishing gradient problem (Goodfellow et al., 2016).

With the vanishing gradient problem, the gradients of neural networks being found using back-propagation, find the derivatives of the network by moving layer-by-layer from the final layer to the initial layer. By the chain rule, the derivatives of each layer are multiplied down the network to compute the derivatives of the initial layer (Russell and Norvig, 2002). However, when n hidden layers use an activation like the sigmoid function, n small derivatives are multiplied together, thus the gradient decreases exponentially as we propagate down to the initial layer creating the vanishing gradient problem that causes training issues within neural networks (Russell and Norvig, 2002). The ReLu function accounts for the vanishing gradient problem as the range of the function is between 0 and infinity which makes this activation function more advanced to other activation functions. Within the ReLu activation

function, negative values are converted to 0 so there are no negative values available therefore removing the vanishing gradient problem so the output prediction accuracy and efficiency is maximized (Glorot et al., 2011).

3.1.2 Neural Network Structures

There are three main categories of neural network structures: acyclic or feed-forward networks, cyclic or recurrent networks, and convolutional neural networks. A feed-forward network represents a function of its current input, therefore it has no internal state other than the weights themselves. A recurrent neural network, feeds its outputs back into its own inputs meaning that the activation levels of the network form a dynamic system that may reach a stable state or exhibit oscillations or even chaotic behavior. Moreover, the response of the network to a given input depends on its initial state, which may depend on previous inputs. Hence, recurrent networks can support short-term memory (Russell and Norvig, 2002). This makes them more interesting as models of the brain, but also are more difficult to understand. Convolutional neural networks allow for input to be used within a convolutional layer which defines a window by which we examine a subset of the input data that subsequentially scans the entire input through this movable window (Strumberger et al., 2019). This window, sometimes called a filter, allows for the scanning of specific features within an input and produces an output which focuses solely on the regions of the image which exhibit the feature it was searching for. Due to this property, convolutional neural networks have gained traction within text analysis along with recurrent neural networks which allow for temporal or time-series input such as text, speech, audio, video, weather and much more.

As previously stated, deep learning approaches using neural networks have gained much popularity in the last 10 years due to their success in improving the state-of-the-art in many fields including NLP. They have shown the capability of modeling complex patterns in large data sets by use of back-propagation which allows the system to automatically find both rules and features needed for classification. In the following sections, we provide a more in-

depth explanation of simple neural networks, recurrent neural networks, and convolutional neural networks.

3.2 Feed-Forward Neural Networks

A feed-forward network represents a function of its current input; thus it has no internal state other than the weights themselves (Krogh, 2008). If we consider the simple neural network shown in Figure 3.2 below which has a_k input units, a_j hidden units, and an a_i output unit. To simplify these expressions, we have omitted the bias units in this example. Given an input vector $x = (x_1, x_2)$, the activation of the input units are set to $(a_1, a_2) = (x_1, x_2)$ and the network computes

$$\begin{aligned} a_5 &= g(W_{3,4}a_3 + W_{4,5}a_4) \\ &= g(W_{3,5}g(W1, 3a_1 + W_{2,3}a_2) + W_{4,5}g(W_{1,4}a_1 + W_{2,4}a_2)) \end{aligned}$$

That is, by expressing the output of each hidden unit as a function of its inputs, the output of the network as a whole, a_5, is a function of the networks inputs. Furthermore, we see that the weights in the network act as parameters of this function; writing W for the parameters, the network computes a function $h_w(x)$. The learning that occurs happens by adjusting the weights therefore changing the function that the network represents. These principles can then be applied to both single-layer feed forward neural networks and multi-layer neural networks (Russell and Norvig, 2002).

Within multi-layer neural networks, the most common case involves a single hidden layer as seen in Figure 3.2. The advantage of adding hidden layers is that it enlarges the space of hypotheses that the network can represent therefore allowing for more complex and abstract computations (Goodfellow et al., 2016). A minor difference in multi-layer and single-layer neural networks is the number of outputs computed. Within multi-layer neural networks we receive several outputs, which provides us with an output vector rather than a single value

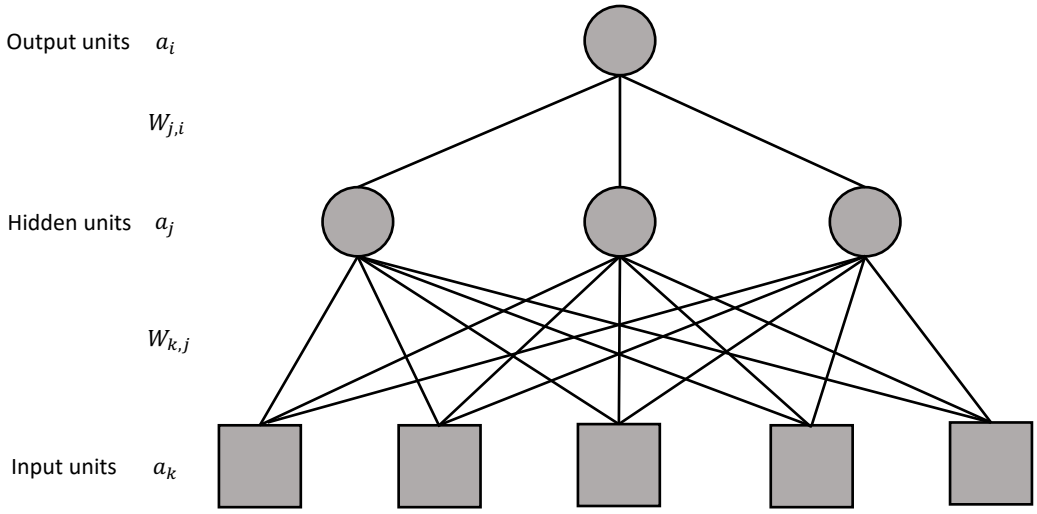

Figure 3.2: *A multi-layer neural network with one hidden layer and 5 inputs.*

(Russell and Norvig, 2002). The major difference is that the error at the output layer is clear but the error at the hidden layers is not clear because the training data does not say what value the hidden nodes should have. To counteract this problem, we can back-propagate the error from the output layer to the hidden layers which emerges directly from a derivation of the overall error gradient (Russell and Norvig, 2002).

At the output layer we have multiple output units so let Err_i be the ith component of the error vector $y - h_W$. We've also found it convenient to define a learning rate α and a modified error $\Delta_i = Err_i \times g'(in_i)$, such that the weight update rule becomes

$$W_{j,i} \leftarrow W_{j,i} + \alpha_j \times \Delta_i$$

To update the connections between the input units and the hidden units, we define a quantity similar to the error term for output nodes. The main idea is that the hidden node j is responsible for some fraction of the error Δ_i in each of the output nodes. Thus, the Δ_i values are divided according to the strength of the connection between the hidden node and the output node and is propagated back to provide the Δ_j values for the hidden layer

(Goodfellow et al., 2016). The propagation rule for the Δ values is the following:

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i.$$

Where g' is the derivative of the activation function. Now the weight update rule for the weights between the inputs and the hidden layer is almost identical to the update rule for the output layer:

$$W_{k,j} \leftarrow W_{k,j} \times \alpha \times a_k \times \Delta_j$$

The back-propagation process can be summarized as follows. We first compute the Δ values for the output units, using the observed error. Starting with the output layer, we repeatedly, for each layer in the network until the earliest hidden layer is reached, propagate the Δ values back to the previous layer and update the weights between the two layers (Russell and Norvig, 2002).

In practice, feed-forward architectures have two main draw backs making them undesirable to be used alone in text classification projects. First, they require a fixed input size which leads to problems in handling real-world samples which often are of variable lengths. Second, the number of parameters in the network increases and this can lead to overfitting with longer samples.

As a result of these disadvantages, the use of fully-connected architectures is often limited to the last layer of the neural networks that are used for text classification (Conneau et al., 2016). To avoid problems mentioned above, two other architectures are often used: recurrent and convolutional neural networks.

3.3 Recurrent Neural Networks

A recurrent neural network (RNN) is composed of recurrent layers that include intra-layer connections in addition to being connected to their previous and next layers. In general, RNNs are capable of processing time-series data, to which natural language texts can belong

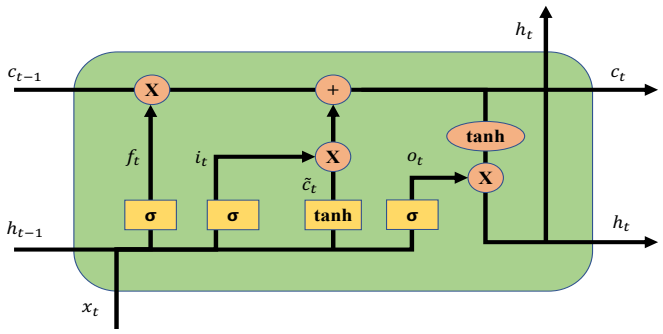

Figure 3.3: *Architecture of an LSTM.*

to. RNN's can be used in forward and backward passes where a forward pass is used when a time-series is fed into the RNN from the first to the last element, while in a backward pass the data is fed from the last to the first.

A vanilla RNN is one of the simplest RNN architectures that have been proposed (Mallya, 2017). The output of a vanilla RNN at time-step t is calculated by using the following equation, where a_t represents the input at time-step t, $h_{t\pm1}$ refers to the output of the vanilla RNN at the previous/next time-steps (based on being in the forward/backward pass), and W represents the weights connecting the input to the RNN unit, and U stands for the weights that connect the RNN unit to itself.

$$h_t = tanh(a_t W + h_{t\pm1} U)$$

During training, vanilla RNNs suffer from the vanishing gradient problem which prevents the network from learning as the sequence of words increases. To avoid these problems, an alternate model called a Long Short-Term Memory (LSTM) model has been proposed.

3.3.1 Long Short-Term Memory

Hochreiter and Schmidhuber (1997) first introduced a gated recurrent architecture called the Long Short-Term Memory (LSTM) by adding *input*, *forget*, and *output* gates to the vanilla RNN. Figure 3.3 shows the achitecture of an LSTM, when being used in a forward pass.

In an LSTM, the values of the input, forget, and output gates at a specific time-step are calculated using the following three equations.

$$i_t = \sigma(a_t W_i + h_{t\pm1} U_i)$$

$$f_t = \sigma(a_t W_f + h_{t\pm1} U_f)$$

$$o_t = \sigma(a_t W_o + h_{t\pm1} U_o)$$

where i_t, f_t, and o_t refer to the values of the input, forget, and output gates at time-step t respectively, x_t represents the input at time-step t, $h_{t\pm1}$ represents the output of the LSTM unit at time-step t, the Ws refer to the weight connecting the input to the gates, the Us represent the connecting weights between the output of LSTM to its gates, and σ refers to the sigmoid activation function.

An LSTM includes a cell state, or memory, whose values are calculated using the following equations.

$$\tilde{C}_t = tanh(a_t W_g + h_{t\pm1} U_g)$$

$$C_t = \sigma(f_t] \times C_{t\pm1} + i_t * \tilde{C}_t)$$

Finally, to provide more information in how the LSTM unit is calculated we use the following equation.

$$h_t = tanh(C_t \times o_t)$$

The advantage of an LSTM model is its ability to handle longer sequences. Having the addition of input, forget, and output gates allows them to be less prone to the vanishing and exploding gradient effect, however, the high number of parameters, or weights, makes them more prone to overfitting.

3.4 Convolutional Neural Networks

Convolutional neural networks (CNN) have commonly been used in practice for image processing by taking advantage of the spatial structure of images. Similar to Tobler's first law of geography which states observations close together are more correlated than those far apart, pixels and their neighbors are assumed to be jointly related so CNNs use feature windows, or filters, to connect the input to one neuron. This operation, called a *convolution*, allows for a smaller number of weights to be shared between inputs which allows for a better capturing of patterns within data.

In recent years CNNs have gained popularity in natural language processing for their computational efficiency and ability to capture generalizations. By applying the convolution concept to natural language, one can assume that the input sections in texts are N-grams, or N consecutive words or characters. This tyoe of neural network can handle the problems of overfitting in feed-forward neural networks and typically can handle inputs of various sizes. In recent years, CNNs have become widely popular because of their use in sentiment analysis.

3.5 Training and Validation of Neural Networks

Roy et al. (2020) propose the use of 9 emotional categories or psychological constructs to convert tweets into emotional classes. Their classes include, anxiety, burdenness, depression, stress, loneliness, hopelessness, and insomnia. The scope of those diagnosed with depressions is quite large, therefore Roy et al. (2020) suggest splitting the depression emotion by date into 3 sub-classes labeled depression 1, depression 2, and depression 3. Depression 1 contains tweets between January 1, 2019 to November 30, 2019. Depression 2 contains tweets between December 1, 2019 to October 31, 2020. And Depression 3 contains tweets between November 1, 2020 to August 31, 2021.

In this chapter, we wish to build upon the concepts previously studied and understand classification using real data. We will present three different types of word embeddings:

Word2Vec, Glove, and FastText pre-trained word-embeddings used in three different types of neural networks designed for classification: Feed-Forward, Convolutional, and Recurrent Long-Short Term Memory neural networks using Keras V. 2.6.0 provided by Chollet et al. (2015).

3.5.1 Neural Network Training Preparation

To begin our emotion classification using neural networks, we first apply a sentiment classification to one emotion, anxiety, to test accuracy metrics for all neural networks. In this case, the data extracted using the anxiety query found in Table 2.2 was assigned a sentiment of 1 and control data extracted using the query, "I" was assigned a sentiment of 0. This is because we wish to train the neural networks to detect anxiety based on patterns found within training. Once a training sentiment was assigned we then check the distribution of sentiments as we wish to obtain approximately equal numbers of positive and negative sentiments and then apply train test split provided by SciKit Learn model selection V. 1.0.2 to split our data into training and testing datasets. With train test split provided by SciKit Learn we are able to change the default change size from 25% to 20% of data to be split into a testing dataset as to follow the most commonly used split in practice. Because we split the data into 20% for testing we receive its complement, 80% for training purposes. Along with specifying the size of testing and training datasets we wish to receive we also specify the random state to be 42 which controls the shuffling applied to the data before applying the split and allows reproducibility among the same results across different calls.

After we have split our data, we must next prepare the embedding layers for our neural networks. To begin, we first apply Keras tokenizer, a separate tokenizer from the regular expression tokenizer found in the Natural Language Tool Kit (NLTK). This tokenizer provides a word-to-index dictionary where each word in the corpus is used as a key, while a corresponding unique index is used as the value for the key. We then apply Keras tokenizer texts to sequence on training and testing data where we receive three list of lists containing integers that map each word in each tweet to its corresponding index found within each type

of word-to-index dictionary. Because tweets vary in lengths, we specify the length of each integer containing list to be equal to the maximum length of all tweets. If the tweet does not reach this number of words, we fill in missing values with a 0 value to obtain equal lengths for each tweet in a process called padding.

Finally, using our three different word embeddings we create our feature matrix. To do this, we first load our pre-trained word embeddings to create a dictionary that contains words as keys and their corresponding embedding list as values. Following word and embedding vector extraction, we will create an embedding matrix where each row number will correspond to the index of the word in the word-to-index dictionary. The matrix produced using Word2Vec pre-trained word embeddings will have 4,803 rows and 300 columns as opposed to Glove and FastText pre-trained word embeddings where we receive 4,803 rows with 100 columns where the number of columns in each of our feature matrices indicates the length of each words embedding vector.

3.5.2 Feed Forward Neural Networks

Once we have pre-processed our text and have obtained our feature matrices we are then able to begin construction of our neural networks. We begin with a simple feed-forward neural network with one hidden layer. To do this, we first create a Sequential model within the Keras library so we are able to build our neural network sequentially, or layer by layer. Next, we create our embedding layer which will have an input length equal to the number of columns in the corresponding word-embedding feature matrix and an output of equal length. We also specify the vocab size which is the number of unique words in the corpus and in this case the number of rows per type of word embedding. Since we are not training our own embeddings, we specify the trainability parameter to be false and the weights parameter we will pass our own corresponding embedding or feature matrix.

After adding the embedding layer to our model, we then directly connect our embedding layer to a densely connected layer meaning each neuron within this layer is connected to all neurons in its previous layer. Within this dense layer we specify our activation function to

be of sigmoid activation which is expressed below.

$$S(x) = \frac{1}{1 + e^{-x}} \tag{3.1}$$

This activation function is the same function used in logistic regression classification algorithms. They work by taking any real value, x, and outputs values within the range of 0 and 1 where the larger the input value the closer our output is to 1. This is especially useful for models where we wish to receive a predicted probability as output.

In order to compile our model, we use the Adam optimizer. This optimizer is an adaptive learning rate optimization algorithm that has been designed specifically for training deep neural networks and is an adaptive learning rate method meaning it computes individual learning rates for different parameters. The name Adam is derived from adaptive moment estimation, because it uses estimations of first and second moments of gradient to adapt the learning rate for each weight of the neural network. In more detail, let m_n be the n-th moment of random variable X defined as the expected value of that variable raised to the power of n.

$$m_n = E[X^n]$$

It is important to note that the gradient of the cost function within the neural network can be considered a random variable as it is usually evaluated on some small random batch of data. The first moment is the mean of the random variable and the second moment is the uncentered variance of the random variable. To estimate these moments, Adam utilizes exponentially moving averages, computed on the gradient evaluated on a current mini-batch.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

In the equations above, m_t and v_t are moving averages that are initialized at 0 upon the first iteration, g is the gradient on the current mini-batch, and β which are new hyper-

parameters of the algorithm. The expected values of the estimators should equal the parameter we are trying to estimate, as it happens, the parameter in our case is also the expected value. If these properties held true, that would mean, that we have unbiased estimators. Because we initialize averages with zeros, the estimators are biased towards zero. The further we go into expanding the value of m_t, the less first values of gradients contribute to the overall value, as they get multiplied by smaller and smaller beta. Upon capturing this pattern we can re-write the formula for our moving average in the equation below.

$$m_t = (1 - \beta_1) \sum_{i=0}^{t} \beta_1^{t-i} g_i$$

Next, the Adam optimizer must correct the estimator in a process called bias correction thus making our formulas for our estimator follow the form of the following equations.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Finally, to complete the Adam optimizer we must use the moving averages to scale the learning rate of each parameter by the following equation where w is the model weights and η is the step size.

$$w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{t}} + \epsilon}$$

Along with specifying the Adam optimizer as our optimizing function in the model compilation, we also specify our loss function to be of Binary Cross-Entropy. This function allows us to measure the difference between two probability distributions for a given random variable or set of events and is used when the output of our model is between classes. Binary Cross Entropy is the negative average of the log of corrected predicted probabilities. By comparing the predicted probabilities to their actual class output this function then calculates a score that penalizes the probabilities based on the distance from the expected value.

$$-\frac{1}{N}\sum_{i=1}^{N}(log(p_i))$$

The equation above shows the equation for calculating the binary cross-entropy of two classes belonging to either 0 or 1 where p_i refers to the individual probability of an observations associated predicted class.

After specifying our loss and optimization functions we then specify our metric and train our model with 6 epochs and a validaton split of 20% such that 20% of our training data will be used to provide our accuracy metric. Once training has been done for our feed forward neural network, we review our results which are displayed in Section 3.6 Results.

3.5.3 Recurrent Neural Network

As discussed previously in Section 3.3 we now prepare a recurrent neural network for emotion classification. Recurrent neural network is a type of neural networks that is proven to work well with sequence data. Since text is actually a sequence of words, a recurrent neural network is an automatic choice to solve text-related problems. In this section, we will use an LSTM (Long Short Term Memory network) which is a variant of RNN, to solve sentiment classification problems.

Once again, we initialize our model with a Sequential model found in the Keras library. We then add our embedding layer similar to that discussed in our feed-forward neural network. After adding those layers to our model we then add a Long Short Term Memory layer with 128 neurons. Upon adding our LSTM layer we then add a Dense layer with sigmoid activation and finally compile it using the Adam optimizer and binary cross-entropy loss function identical to our feed forward neural network. Finally, we train our model specifying our model to run 6 epochs and a validation split of 20% such that 20% of our training data will be used to provide our accuracy metric. Once trianing has been done for our recurrent neural nework, we review our results which are displayed in Section 3.6 Results.

3.5.4 Convolutional Neural Network

As discussed in Section 3.4 a convolutional neural network is a type of network that is primarily used for 2D data classification, such as images. This type of network tries to find specific features in an image in the first layer and in the next layers, the initially detected features are joined together to form bigger features. In this way, the whole image is detected. Although convolutional neural networks have been proven successful in image detection, they also work well with textual data. Though text data is one-dimensional, we can use 1D convolutional neural networks to extract features from our data.

To begin, similarly to a feed-forward neural network, we first initialize a Sequential model using the Keras library. We then establish our embedding layer identical to our feed forward neural network. Following initialization and establishing our embedding layer, we then declare a one-dimensional convolutional layer with 128 features and our kernel size of 5. This kernel size allows for 5 individual observations to be viewed simultaneously by use of a rolling window. Within this convolutional layer we also specify the convolutional activation function to be the Rectified Linear Unit (ReLU) whose equation is seen below.

$$y = max(0, x) \tag{3.2}$$

The above function works by finding the maximum value between 0 and a random variable x. As opposed to deep layers using nonlinear activation functions such as the sigmoid function fail to receive useful gradient information since error is back propagated through the network and used to update the weights. The amount of error decreases dramatically with each additional layer through which it is propagated, given the derivative of the chosen activation function. This is called the vanishing gradient problem and prevents deep (multi-layered) networks from learning effectively. In order to use stochastic gradient descent with backpropagation of errors to train deep neural networks, an activation function is needed that looks and acts like a linear function, but is, in fact, a nonlinear function allowing complex relationships in the data to be learned.

ReLU grants the neural network with three advantages: computational simplicity, repre-

sentational sparsity or its ability to output a true 0 value, and linear behavior such that the function both looks and acts like a linear function.

After specifying our convolutional layer activation function to be ReLU we then move into a global max pooling layer. This layer finds the maximum value for each feature map or rolling window within the previous convolutional layer. This global pooling layer can be used in a variety of cases. Primarily, it can be used to reduce the dimensionality of the feature maps output by some convolutional layer, to replace Flattening and sometimes even Dense layers in your classifier (Christlein et al., 2019). What's more, it can also be used for word spotting due to the property that it allows detecting noise, and thus "large outputs" (Sudholt and Fink, 2016).

Finally, within the convolutional neural network we specify a dense layer with sigmoid activation and compile it using the Adam optimizer and binary cross-entropy as our loss function similar to that in the feed-forward neural network. We then specify our metric and train our model with 6 epochs and a validaton split of 20% such that 20% of our training data will be used to provide our accuracy metric. Once training has been done for our convolutional neural network, we review our results which are displayed in Section 3.6 Results.

3.5.5 TextBlob

Further variables were introduced in the Roy et al. (2020) analysis which include subjectivity and polarity provided by the TextBlob module version 0.17.1. TextBlob provides a simple API for diving into common natural language processing tasks such as parts-of-speech tagging, noun phrase extraction, sentiment analysis, and many more tasks. TextBlob subjectivity lies within the bounds of [0,1] and quantifies the amount of personal opinion and factual information contained in the text. The higher subjectivity means that the text contains personal opinion rather than factual information. The polarity score within TextBlob lies within [-1,1] where -1 defines a negative sentiment and 1 defines a positive sentiment.

With the use of TextBlob, each tweet was classified with a subjectivity and polarity score

and was joined to the emotion classifications to provide us with 11 variables of interest within our dataset.

3.6 Results

After extracting our tweets, pre-processing each tweet, building and compiling the aforementioned models for each type of pre-trained word embedding, we now compare the accuracy of each anxiety model. Because we are using the neural networks to classify emotions, similar to a sentiment analysis, each psychological constructs neural network should receive similar accuracy scores. Therefore, the loss, accuracy, and ROC curves for each type of word embedding and anxiety model is shown below in Figure 3.4. The remaining psychological construct figures are supplied in Appendix A, Supplementary Figures.

As seen in Figure 3.4, the pre-trained FastText embedding layer performs the worst. This could be due to the training of unknown word embeddings. Because some words may be misspelled or may not exist within the FastText pre-trained embeddings, we train the new words with the words around it to form an embedding vector specific to that word. One can deduce that because of the training done with the FastText embedding, the most recent embedding vector for a certain word was used within the embedding matrix even though the trained vector only applies to one specific tweet. Further, the number of neurons or features set within the LSTM model for all word embeddings could be parameterized better to include all available features instead of the stated and fixed 128 initial neurons within all LSTM models.

Also seen in Figure 3.4, the simple neural network, although performed well did not perform better than the convolutional neural network with accuracy scores of Word2Vec and Glove accuracy of 0.786 and 0.787 respectively for the anxiety psychological construct model. It comes to no surprise that Word2Vec and Glove embeddings performed the best within the convolutional neural network with accuracy scores of 0.96 and 0.94 respectively. This aligns with other articles regarding text analysis such as Roy et al. (2020) who use a convolutional neural network for text classification purposes.

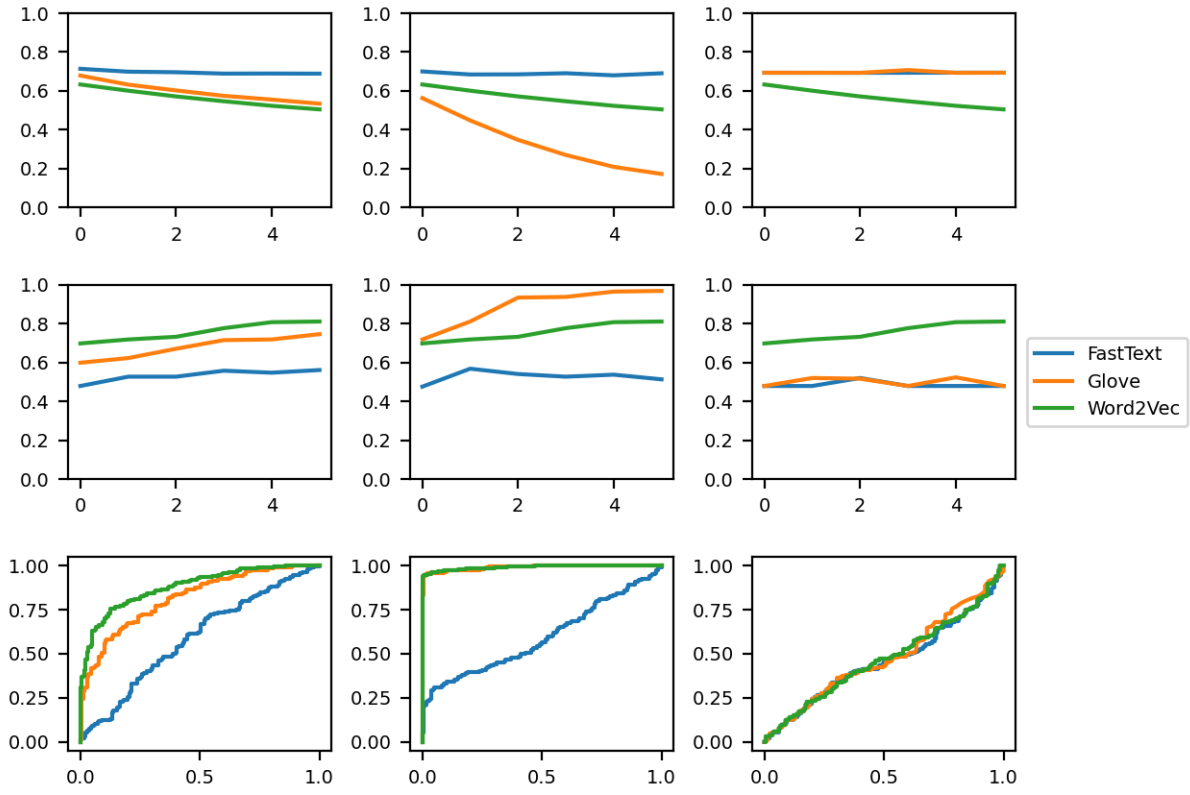

Figure 3.4: *Three neural network and word embedding performance for the psychological construct anxiety. Three metrics loss* `(top row)`*, accuracy* `(middle row)`*, and ROC* `(bottom row)` *for models SNN* `(left)`*, CNN* `(center)`*, and LSTM* `(right)`*. Each line represents the word embedding methods: FastText* `(blue)`*, Glove* `(yellow)`*, and Word2Vec* `(green)`*.*

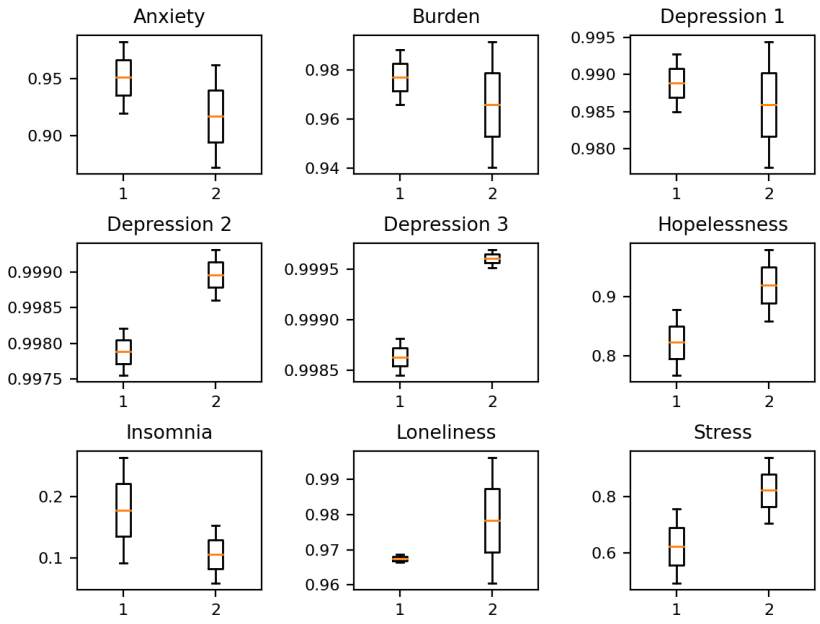

Figure 3.5: *Boxplots comparing cases (1) and controls (2). Anxiety, Burden, Hopelessness, Insomnia, and Stress provide the largest difference when comparing the average of each psychological constructs cases and controls which provides evidence to support the claim that these constructs will provide the most information to the random forest.*

With our results over the best pre-trained word embedding and neural network, we now are able to make predictions per psychological construct. Each pscyhological construct was trained using the Glove word embedding as to align with most current text analysis literature.

Using the trained psychological construct neural networks, we were then able to make predictions. By tokenizing the tweets within the data set that contains the timelines using the NLTK Regular Expression Tokenizer and the TensorFlow Tokenizer, we then flatten each tweet and add padding to each tweet to reach the required input length. We then used the predict function to predict the score for each tweet per psychological construct.

To elaborate on the results of the models predictions, we compare each psychological constructs box plots as shown in Figure 3.5. We can see that Anxiety, Burden, Hopelessness, Insomnia and Stress show the largest difference between the cases and controls. In anxiety we see that those classified as cases have an average anxiety value of 0.95 whereas those classified as controls have an average value of 0.92 with a difference of 0.03 between the cases

and controls. In the Burden construct we see that those classified as cases have an average burden value of 0.97 whereas those classified as controls have an average burden value of 0.96 with a difference of 0.01 between the cases and controls. Hopelessness cases have a lower average of 0.81 where as controls have an average of 0.91 with a difference between cases and controls of 0.10. The stress construct has the largest difference between cases and controls with an average case value of 0.61 and control average of 0.81 with a difference of 0.20. We would expect the case values to be relatively higher among all psychological constructs but Depression 2, Depression 3, Hopelessness, Loneliness, and Stress all have larger control values. Though some constructs do not exhibit significant differences, the random forest algorithm is able to detect these differences and classify accordingly by applying certain constraints for branching.

To further elaborate on the interpretation of the models prediction results, we randomly chose two tweets. One tweet marked as a case for suicidal ideation,"i always think suicide is key but i am just waiting to prove that i am wrong", and one control, "life is good." We provided their scores for each psychological construct below in Table 3.6 and plotted the tweets psychological constructs in Figure 3.6. As seen in Table 3.6, the values returned are between 0 and 1 because we mapped the positive outputs to 1 and the negative outputs to 0. However, the sigmoid function present within the dense layer, provides us with a floating value between 0 and 1.

Referring to Table 3.6, we can see that a strong positive classification for the aforementioned suicidal tweet includes all constructs except insomnia which is classified as a negative and the control tweet shows similar results but are classified slightly more up-regulated in anxiety, hopelessness, stress, insomnia, subjectivity, and polarity constructs and down-regulated among the burden, depression 2, and depression 3 constructs.

Construct	Case Score	Control Score
Anxiety	0.83735	0.90845
Hopelessness	0.75637	0.88329
Burden	0.90417	0.81471
Loneliness	0.96071	0.96615
Stress	0.81799	0.87319
Insomnia	0.17616	0.18729
Depression 1	0.97709	0.97829
Depression 2	0.99754	0.78198
Depression 3	0.99909	0.81732
Subjectivity	0.95000	0.59982
Polarity	-0.25000	0.64293

Table 3.1: *Psychological construct scores for one randomly selected case tweet, "i always think suicide is key but i am just waiting to prove that i am wrong" and one randomly selected control tweet, "life is good".*

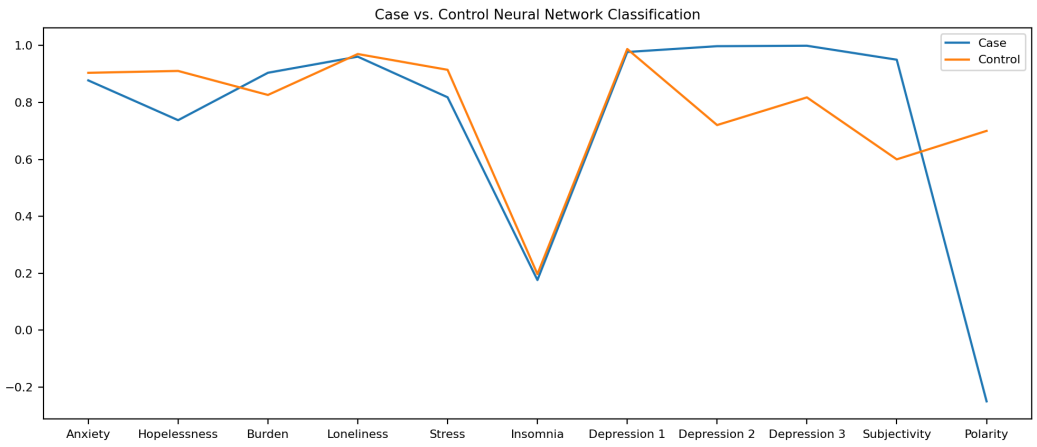

Figure 3.6: *Plotting of the psychological constructs along the x-axis for one randomly selected case tweet, "i always think suicide is key but i am just waiting to prove that i am wrong" and one randomly selected control tweet, "life is good".*

Chapter 4

Suicidal Ideation Risk Assessment

In recent years, online textual data has been used to predict eating disorders and provide mental health assessments. In this chapter, we present how the Random Forest method can be used to predict suicidal ideation within 4, 7, 14, and 21 days using the emotional classification data calculated in Chapter **??**. After tweet extraction we classified each tweet using the neural networks discussed previously, these predicted outcomes can be used to classify an ensemble of decision trees that have high accuracy and precision when calculating the risk of suicidal ideation for a user in the near future.

4.1 Decision Trees and Random Forests

Decision trees and random forests have become more prevalent in programs designed for classification and regression. Developed by Breiman (2001), this algorithm has the ability to handle high dimensional data, can ignore irrelevant descriptors, and is easy to train and interpret. Similar algorithms include, support vector machines (SVM), artificial neural networks (ANN), partial least squares (PLS), k-nearest neighbors (KNN), multiple linear regression (MLR), and linear discriminate analysis (LDA).

Although all the methods previously mentioned allow us to perform classification and regression, they struggle with prediction accuracy and high dimensionality. For example,

KNN and ANN are not efficient with high dimensional data without the use of dimension reduction or the pre-selection of descriptors. Although, SVM does allow for high dimensionality, it suffers from the presence of irrelevant descriptors, thus descriptor pre-selection must be performed. Linear methods such as LDA, MLR and PLS can only handle data where the number of predictors is smaller than the number of descriptors unless pre-selection is again used for dimension reduction.

Therefore, an algorithm that is able to ignore irrelevant descriptors without the need for dimension reduction is needed to perform text classification with high prediction accuracy.

4.1.1 Decision Trees

Decision trees (DT) are one of the most successful classification and regression algorithms available due to its simplicity, comprehensibility, limited number of parameters, and its ability to handle mixed data (Charbuty and Abdulazeez, 2021). DTs are a successive model that compares numeric features to a threshold value in each test which makes it much easier to construct conceptual rules rather than numeric rules like those used in other classification methods (Svetnik et al., 2003).

Nodes and branches that compose a DT provide a method for the grouping of data based on rules computed by the algorithm. Nodes represent features used to split the data and each subset is defined by a value or values of the node feature. Although, nodes and branches that help in classification are informative and flexible, data must have a low level of randomnness or impurity so that decision trees can be grown, otherwise the tree will not branch and separate the data into distinct classes.

Information gain is one metric used for segmentation and is often called, "mutual information" (Charbuty and Abdulazeez, 2021). This informs how much knowledge a random variable's value provides. Using the following equation, we are able to calculate the Gain(S,A) of a variable by utilizing the Entropy equation to calculate the information gain (Charbuty and Abdulazeez, 2021).

$$Gain = E_{parent} - E_{child}$$

Where E_{parent} represents the entropy of the parent node and E_{child} represents the average entropy of the child nodes. Furthermore, entropy can be calculated by utilizing the proportion, $\hat{p}_{m,k}$, of observations in the m^{th} region that are from the k^{th} class and is calculated using the following equation.

$$Entropy = -\sum_{k=1}^{k} \hat{p}_{m.k} log(\hat{p}_{m,k})$$

Gini Impurity is a measurement used to build Decision Trees to determine how the features of a dataset should split nodes to form the tree. More precisely, the Gini Impurity of a dataset is a number between 0-0.5, which indicates the likelihood of new, random data being misclassified if it were given a random class label according to the class distribution in the dataset. If we have K classes and $p(i)$ is the probability of picking a datapoint with class i, then the gini impurity is calculated as the following eqauation.

$$GiniImpurity = 1 - \sum_{i=1}^{K} p_i^2 \tag{4.1}$$

Becuase the Gini impurity is a measure of anti-homogeneity, the feature with the lowest Gini impurity is selected to be the best split feature.

DTs simplicity and interpretability makes them appealing for classification purposes. Moreover, they are able to handle high-dimensional data. However, they do have a major drawback, they usually have relatively low prediction accuracy or high variability. A small change in the data may produce a different result. To overcome DTs problems, Breiman et. al. propose using an ensemble of decision trees called a Random Forest. (Breiman, 2001).

4.1.2 Random Forests

Random forests as proposed by Breiman (2001) are an ensemble of decision trees that are aggregated to provide a higher prediction accuracy but still allow for high-dimensional data

(Breiman, 2001). To further elaborate, the authors suggest using bagging methods to generate training data subsets for building individual trees and randomly selecting a subspace of features at each node to grow branches of decision trees where we then combine all trees to form a random forest model (Xu et al., 2012). To understand the bagging method, one must first understand bootstrapping. In essence, bootstrapping is random sampling with replacement from the available training data set. Bagging, otherwise known as bootstrap aggregation, is performing bootstrapping many times and training an estimator for each bootstrapped sample (Lohr, 2021). Since textual data has many uninformative features to support a specific class or topic without the use of bagging during the forest algorithm, topic-related or informative features would have a large chance of being missed.

Decision trees usually suffer from overfitting, a random forest seeks to minimize overfitting the data by creating a subset of the original data to apply the bagging method. By doing this it provides a higher chance of selecting different data points to the model (Strobl et al., 2009). Unlike boosting, which is a sequential process where each model tries to correct the errors of its previous model, bagging allows the random forest to apply subsets to different decision trees and then calculates the average ranking therefore reducing the risk of overfitting which provides great flexibility for regression and classification problems.

4.2 Suicidal Ideation User Classification

Before applying a random forest method to predict suicidal ideation, we first must identify individual users who show signs of suicidal ideation. For a more accurate representation of suicidal ideation users, a professionally licensed psychiatrist would read through each individual users timeline and classify tweets where the individual shows signs of suicidal ideation. Signs of someone struggling with suicidal ideation include untreated mental disorders (i.e. schizophrenia, anxiety disorders, and certain personality disorders), alcohol or substance abuse, impulsive or aggressive tendencies, feelings of hopelessness, history of trauma or abuse, major physical illness, previous suicide attempts, family history of suicide, and job or financial loss (National Suicide Prevention Lifeline, 2021). Due to the lack of available

resources, all suicidal ideation tweets were classified without a licensed psychiatrist therefore bias may be introduced due to lack of proper training in psychology.

Upon classifying suicidal ideation for training, 342 tweets were found to show signs of suicidal ideation. These tweets included references to self harm, negative self-image, repeated thoughts of suicide, and previous suicide attempts. Tweets ranging from the identified suicidal tweet to tweets ranging 4, 7, 14, and 21 days in the past were saved to a dataframe to be aggregated for single observations reducing the variance associated with each account associated with suicidal thoughts. According to a cross-sectional study provided by Nock et al. 9.2% of the general population suffers from suicidal ideation (Nock et al., 2008). Therefore, controls ranging within the same time frame were aggregated to give a single vector and provided 90% of the data that will be used within future suicidal ideation classification.

4.3 Suicidal Ideation Prediction

After classifying suicidal tweets, aggregating by number of days per suicidal tweet and control tweet, we first split the data into 3 datasets: training, validation, and evaluation using scikit-learn version 1.0. We applied a 33% split where 33% of the data would be used for testing and the rest used for training.

We then apply several Random Forest algorithms to assess and predict future suicidal ideation. For each Random Forest, we specify 100 decision trees within each forest and apply a commonly used random state of 42. We apply this random state because Random Forests use the bagging method applied to decision trees and we need random numbers to select the random samples on which trees are fitted. This implies that each time you generate a set of random numbers the program will generate a completely different set which impacts your samples and in turn the fitted trees. To control the stochasticity involved in random number generation and to replicate the same set of random numbers every time we use a random seed. The random state parameter allows you to set a random seed and fix your random number generation process in a random forest. Therefore, if we run the model again with the same data, we should receive the same output. Since we are assessing several ranges of

aggregated days, we use four Random Forests to model future risk of suicidal ideation such that each set of aggregated days gets its own Random Forest algorithm. Within the next section, we present the results of our four Random Forest models.

4.4 Results

In this section, we present the results of each Random Forest model. We provide accuracy metrics and each models corresponding confusion matrix. We also provide the ROC curve of each model and the importance of each feature in the model.

4.4.1 Random Forest Scores

Using equations 1.1-1.5, we calculated the accuracy measurements, precision measurements, recall measurements, and F1 Score for each corresponding Random Forest Model.

Method	Accuracy	Precision	Recall	F1
4 Days	0.9649	1.0000	0.6735	0.8049
7 Days	0.9737	0.9600	0.6857	0.7999
14 Days	0.9627	1.0000	0.5952	0.7463
21 Days	0.9737	1.0000	0.7447	0.8537

Table 4.1: *Random Forest Metrics using 4, 7, 14, and 21 days worth of aggregated data.*

As seen in Table 4.1, each Random Forest model has a relatively high accuracy. Though accuracy provides a metric describing the number of correctly classified data instances over the total number of data instances it does not provide a good metric of false negative and false positives. Recall, or the measurement of false negatives, is more important than both the accuracy and precision due to the diagnostic needs of our model. Within Table 4.1, the highest recall score lies within 21 days worth of aggregated data. With a recall score of 0.7447, the possibility of false negatives is lowest in comparison to the three other models methods with recall rates ranging within 0.5 and 0.8.

Using Figure 4.1, we see that 21 days worth of aggregated data provides the highest ROC rate within the model. We can also see that using 4, 7, and 14 days worth of aggregated data

provides a significantly lower ROC rate due to the increase in variance within the data. This variance is reduced by the aggregation of more data within the model therefore, providing more accurate predictions.

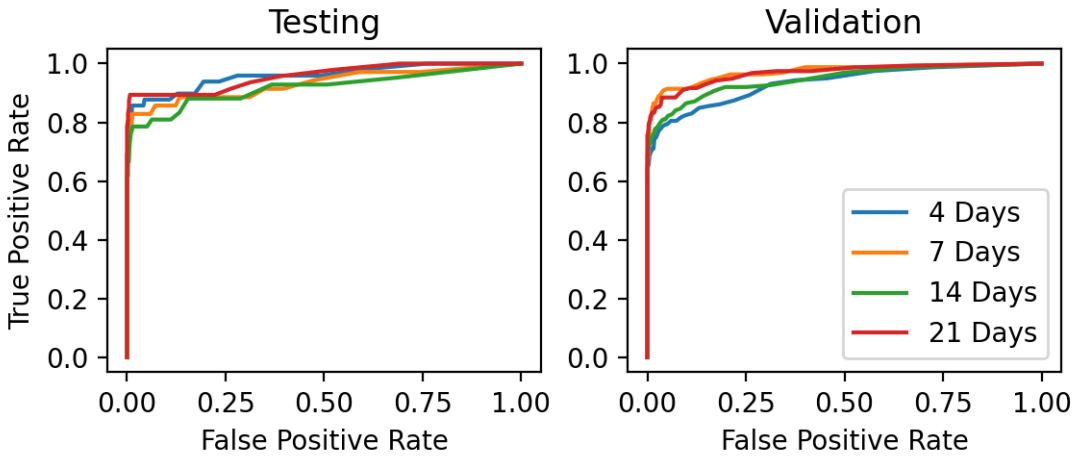

Figure 4.1: *ROC curves for testing dataset and validation set for 4 days* `(blue)`*, 7 days* `(yellow)`*, 14 days* `(green)`*, and 21 days* `(red)` *worth of aggregated data.*

Using SciKit learns Random Forest implementation we also receive the feature importance of each psychological construct. This feature importance is calculated using Equation 4.1 which calculates each features importance as the sum over the number of splits across all trees that include the feature and the proportionality to the number of samples it splits. The feature importance matrix helps in visualizing the importance of each feature across 4, 7, 14, and 21 days. Figure 4.2 below provides a visualization of our feature importance matrix shown in Table 4.2.

As seen in Table 4.2, Depression 1 and subjectivity provide a consistently high Gini impurity. The feature Depression 3 provides a low Gini impurity across all models except our 14 and 21 Days models where Depression 3 moves from a low Gini impurity to higher values. Features such as Burden, Insomnia, Hopelessness and Loneliness vary within each model. Although most of these features appear to have a low Gini impurity, their Gini impurity calculation vary across all models ranging from a mid-level importance to low importance.

Features	4 Days	7 Days	14 Days	21 Days
Anxiety	0.077409	0.067433	0.083585	0.081481
Burden	0.070314	0.073444	0.085457	0.099878
Depression 1	**0.116956**	**0.125427**	**0.106891**	**0.124485**
Depression 2	0.085272	0.079699	0.094893	0.094092
Depression 3	0.096618	0.101365	**0.134327**	**0.122664**
Hopelessness	0.064361	0.068267	0.074574	0.073951
Insomnia	0.080945	0.096030	0.085803	0.091073
Loneliness	0.074216	0.082864	0.092528	0.066623
Polarity	**0.113331**	**0.104181**	0.058444	0.065403
Stress	0.085808	0.087324	0.080601	0.066265
Subjectivity	**0.134769**	**0.113967**	**0.102896**	**0.114137**

Table 4.2: *Feature importance of Random Forest algorithm. With 4 days of aggregated data Depression 1, Depression 3, and Subjectivity supply the most information for splitting. With 7 days of aggregated data Depression 3, Loneliness, and Subjectivity supply the most information for splitting. With 14 days of aggregated data Depression 3, Hopelessness, and Subjectivity supply the most information for splitting. And with 21 days of aggregated data Depression 3, Loneliness, and Subjectivity supply the most information for splitting.*

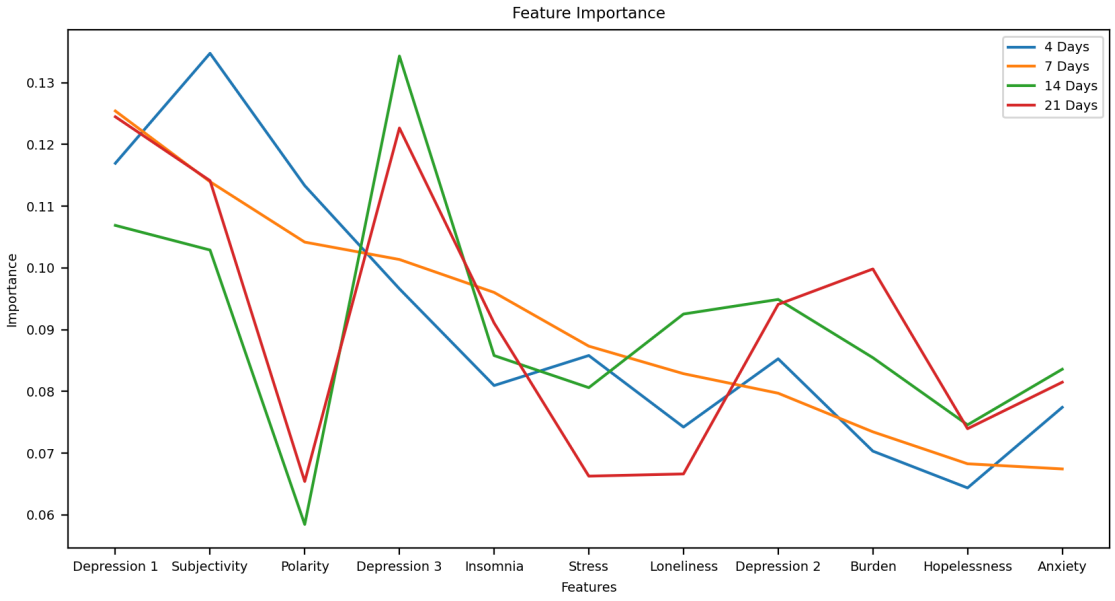

Figure 4.2: *Random Forest feature importance for 4, 7, 14, and 21 days worth of aggregated data.*

Chapter 5

Discussion and Conclusion

In this chapter we summarize the conclusions of the several steps of the analysis we presented in this report. In addition, we will discuss some aspects of the models that became evident during implementation, a discussion of the methods used within this report and a conclusion to this report, future work within the scope of this research, and suggestions of other methods that could be applied to reach a Twitter users suicidal ideation classification with better accuracy and recall to better aid in the prevention of suicides.

5.1 Discussion

Within this section we wish to compare our results with the results of Roy et al. (2020) and review issues that can be encountered when fitting these models. We also wish to elaborate on other possible techniques that could be used to reach the same goal of suicidal ideation classification alongside future work within the scope of this research.

To begin our discussion, by reviewing the Roy et al. (2020) results. They analyzed 512,526 tweets from N = 283 cases and 3,518,494 tweets from 2655 controls and use neural networks and sentiment polarity to generate an array of ten metrics for each tweet. Within their constructed neural networks they obtained an AUC above 70% for each neural network model. This is in line with the work that has been presented in this report. Using convo-

lutional neural networks and Glove pre-trained word embeddings, we observed AUC scores above 70% for all psychological constructs. Though, with these high AUC scores, questions arise due to the changes in common language throughout the years. Within 2019 alone, the Oxford English Dictionary added no fewer than 650 new words, phrases, and senses to their dictionary (oxf, 2019). With the addition of new words and phrases added to the English dictionary, pre-trained word embeddings will become less valuable as new words may appear within a users Twitter Timeline that may not be present in pre-trained word embeddings. Along with inaccurate pre-trained word embeddings, training for neural networks must be done periodically throughout the years. Because of the continuous changes and additions to the English dictionary, new words that are not present within the pre-trained word embeddings or inaccurate word embeddings will provide higher bias and variance within each psychological construct. With higher variance and bias within the psycholgical constructs, the prediction of suicidal ideation may become less accurate and pose a risk to those categorized as false negatives, otherwise known those who are incorrectly classified as having suicidal ideation when in fact provide evidence for an increased risk to suicide.

Within the Random Forest models that predict suicidal ideation, Roy et al. (2020) obtain an AUC of 88% which is comparable to the 90% AUC score obtained within this report. Although the AUC for our 7 day Random Forest model is slightly lower than the Roy et al. (2020) obtained AUC score, the results hold steady through multiple evaluations of our Random Forest model. In Section 4.3 we present the parameter, random state, that is used to set a random seed so that reproducibility of the results may be obtained. By changing this seed, we obtain relatively similar results with AUC scores above 85% per iteration of the Random Forest method. This provides evidence that results using the obtained data are not randomly obtained, thus providing evidence of the functionality of suicidal ideation classification.

Roy et al. (2020) also provide geographic based analysis which was not applied within this report. This is due to the settings each user has within their own Twitter account where each user has the option of providing a geographic location or georeference with each tweet they post. Because most users do not allow for this function, we were not able to analyze

geographic location. This may present an issue within our analysis by the introduction of bias and an increase of variance due to the assumption that each geographic location is the same as those far away. Using Tobler's First Law of Geography, which states that observations close together are more related than those that are further away, we can deduce that people located in California may have different behavioural influences than those located in New York (Miller, 2004). Different results within the model may be obtained when geographic location is involved as a feature within this report.

Roy et al. (2020) also provide an age based analysis which was not used within the scope of this research. This analysis may provide more information regarding suicidal ideation and the risk of suicide based on age. Though Twitter's Academic Access provides more valuable information to researchers, it does not provide the age of each Twitter user. It is assumed that each Twitter user is above the age of 13 but younger users may be present since there is limited monitoring of age among Twitter users. We did not include this due to the lack of information within the dataset and because users have the ability to lie within their agreement with Twitter and their requirement that you are at least 13 years old. Though Twitter accounts are monitored, without posting your age or photos of yourself revealing your age, Twitter Academic Access has limited evidence to support a fraudulent Twitter usage agreement to suspend or eliminate the users account. Along with the falsity of age and Twitter's agreement, we also do not know if each Twitter account is associated with only one user. One Twitter account may be run by more than one user and therefore may provide inaccurate results following the idea that each Twitter account is only associated with one user. Since we do not know if a Twitter account is shared between users, we assume that each Twitter account is its own independent experimental unit as opposed to assuming that each individual user is the experimental unit within the scope of this research. Along with the analysis of age and independent Twitter accounts, Roy et al. (2020) also provide evidence to suggest that gender may provide sufficient evidence to help classify suicidal ideation and risk.

Roy et al. (2020) do not provide sufficient details about some important elements, therefore certain assumptions were made to reproduce their work. The number and types of layers

in neural networks were not described. It is also unclear what type of word embedding they use. In this report we use pre-trained word embeddings but within the research by Roy et al. (2020) we are not explicitly told what type of word embedding they use within their research and if they used pre-trained word embeddings, trained their own word embeddings, or a combination. Within this report we primarily focus on pre-trained word embeddings obtained from Facebook, Google, or Stanford though other pre-trained word embeddings have been obtained such as TwitterEmbeddings which was created using words found specifically within Twitter accounts and embedding vectors were calculated using Twitter data. TwitterEmbeddings may provide a better vector representation for words used within Twitter as opposed to Glove's pre-trained word embeddings.

The last aspect to consider is the choice of a Random Forest classifier. Here we stayed within the self-imposed restriction of emulating the Roy et al. (2020) approach. However, other methods may be of interest to obtain the same goal. For example, boosting methods such as AdaBoost or XGBoost may provide better results through their iteration and constant updates of weights to reduce error until a relatively low error rate is obtained and because AdaBoost and XGBoost libraries are primarily built to build decision trees, boosting methods may be of interest for future work within the scope of this study without the need for new types of data. Although with our models, there is evidence to support that boosting methods may not be necessary with these models as we receive relatively decent accuracy, recall, and precision metrics. Support Vector Machines may also provide results similar to a Random Forest method though data manipulation and transformations may need to be done for the Support Vector Machine to classify suicidal ideation correctly. Naive Bayesian classifiers and Logistic Regression classifiers may also be used with similar data manipulations or transformations for these methods to be used (Shah et al., 2020; Troussas et al., 2013).

5.2 Conclusion

Within this report we presented a detailed explanation of some of the methods used within the Roy et al. (2020) study. We also test the reproducibility of their work by training 9 independent neural networks to obtain classifications of psychological constructs associated with suicide and suicidal ideation including burden, insomnia, stress, anxiety, depression, hopelessness, and loneliness. With AUC's above 75% per psychological construct neural network, we then use the array produced to predict the classification of suicidal ideation among Twitter accounts. Within their research, Roy et al. (2020) obtain an AUC of 88% with a Random Forest method to classify suicidal ideation among Twitter users which is slightly higher than the 83% AUC score obtained within this report following the Roy et al. (2020) study. Though this research provides results similar to those found in Roy et al. (2020) it is of interest to note possible areas of concern about this work. The continuous training of neural networks throughout the years may provide different results in the future when classifying Twitter accounts with suicidal ideation and the addition of new words may pose issues for the future of this methodology. Though issues regarding the usage of these types of models may be more trivial than some other well established models for sentiment analysis such as sentiment analysis using deep learning neural networks or Naive Bayes methods, this research does provide a new direction of statistical analysis in which statistics and the collaboration among professions play an increasingly important part in the future of statistical analyses with the goal to provide better tools for classification of suicidal individuals as opposed to the limited resources available currently.

Bibliography

New words in the oed: March 2019, Oct 2019. URL https://public.oed.com/blog/new-words-in-the-oed-march-2019/.

S. S. Abdullah, M. S. Rahaman, and M. S. Rahman. Analysis of stock market using text mining and natural language processing. In *2013 International Conference on Informatics, Electronics and Vision (ICIEV)*, pages 1–6. IEEE, 2013.

A. Al-Khatib and S. R. El-Beltagy. Emotional tone detection in arabic tweets. In *International Conference on Computational Linguistics and Intelligent Text Processing*, pages 105–114. Springer, 2017.

M. F. Ballesteros, S. A. Sumner, R. Law, A. Wolkin, and C. Jones. Advancing injury and violence prevention through data science. *Journal of safety research*, 73:189–193, 2020.

Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

D. S. Carrell, D. Cronkite, R. E. Palmer, K. Saunders, D. E. Gross, E. T. Masters, T. R. Hylan, and M. Von Korff. Using natural language processing to identify problem usage of prescription opioids. *International journal of medical informatics*, 84(12):1057–1064, 2015.

W. W. Chapman, J. N. Dowling, O. Ivanov, P. H. Gesteland, R. Olszewski, J. U. Espino, and M. M. Wagner. Evaluating natural language processing applications applied to outbreak and disease surveillance. In *Proceedings of 36th symposium on the interface: computing science and statistics*, volume 2004. Citeseer, 2004.

B. Charbuty and A. Abdulazeez. Classification based on decision tree algorithm for machine learning. *Journal of Applied Science and Technology Trends*, 2(01):20–28, 2021.

F. Chollet et al. Keras, 2015. URL https://github.com/fchollet/keras.

V. Christlein, L. Spranger, M. Seuret, A. Nicolaou, P. Král, and A. Maier. Deep generalized max pooling. In *2019 International conference on document analysis and recognition (ICDAR)*, pages 1090–1096. IEEE, 2019.

F. Colas and P. Brazdil. Comparison of svm and some older classification algorithms in text classification tasks. In *IFIP International Conference on Artificial Intelligence in Theory and Practice*, pages 169–178. Springer, 2006.

A. Conneau, H. Schwenk, L. Barrault, and Y. Lecun. Very deep convolutional networks for text classification. *arXiv preprint arXiv:1606.01781*, 2016.

L. Dini and A. Bittar. Emotion analysis on twitter: the hidden challenge. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 3953–3958, 2016.

A. Ebadi, P. Xi, S. Tremblay, B. Spencer, R. Pall, and A. Wong. Understanding the temporal evolution of covid-19 research through machine learning and natural language processing. *Scientometrics*, 126(1):725–739, 2021.

X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

Y. Goldberg. Neural network methods for natural language processing. *Synthesis lectures on human language technologies*, 10(1):1–309, 2017.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

J. Hirschberg and C. D. Manning. Advances in natural language processing. *Science*, 349 (6245):261–266, 2015.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

G. Jacobs, E. Lefever, and V. Hoste. Economic event detection in company-specific news text. In *Proceedings of the First Workshop on Economics and Natural Language Processing*, pages 1–10, 2018.

G. James, D. Witten, T. Hastie, and R. Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov. Fasttext. zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016a.

A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016b.

H. Khanpour and C. Caragea. Fine-grained emotion detection in health-related online posts. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1160–1166, 2018.

A. Krogh. What are artificial neural networks? *Nature biotechnology*, 26(2):195–197, 2008.

A. Le Glaz, Y. Haralambous, D.-H. Kim-Dufor, P. Lenca, R. Billot, T. C. Ryan, J. Marsh, J. Devylder, M. Walter, S. Berrouiguet, et al. Machine learning and natural language processing in mental health: Systematic review. *Journal of Medical Internet Research*, 23 (5):e15708, 2021.

R. Leaman, L. Wojtulewicz, R. Sullivan, A. Skariah, J. Yang, and G. Gonzalez. Towards internet-age pharmacovigilance: extracting adverse drug reactions from user posts in health-related social networks. In *Proceedings of the 2010 workshop on biomedical natural language processing*, pages 117–125, 2010.

S. L. Lohr. *Sampling: design and analysis*. Chapman and Hall/CRC, 2021.

S. Madisetty and M. S. Desarkar. An ensemble based method for predicting emotion intensity of tweets. In *International Conference on Mining Intelligence and Knowledge Exploration*, pages 359–370. Springer, 2017.

A. Mallya. Some rnn variants, 2017.

W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch, and A. Joulin. Advances in pre-training distributed word representations. *arXiv preprint arXiv:1712.09405*, 2017.

H. J. Miller. Tobler's first law and spatial analysis. *Annals of the association of American geographers*, 94(2):284–289, 2004.

National Suicide Prevention Lifeline. We can all prevent suicide, 2021. URL https://suicidepreventionlifeline.org/how-we-can-all-prevent-suicide/.

M. K. Nock, G. Borges, E. J. Bromet, J. Alonso, M. Angermeyer, A. Beautrais, R. Bruffaerts, W. T. Chiu, G. De Girolamo, S. Gluzman, et al. Cross-national prevalence and risk factors for suicidal ideation, plans and attempts. *The British journal of psychiatry*, 192(2):98–105, 2008.

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

J. Pestian, H. Nasrallah, P. Matykiewicz, A. Bennett, and A. Leenaars. Suicide note classification using natural language processing: A content analysis. *Biomedical informatics insights*, 3:BII–S4706, 2010.

S. Poria, N. Majumder, R. Mihalcea, and E. Hovy. Emotion recognition in conversation: Research challenges, datasets, and recent advances. *IEEE Access*, 7:100943–100953, 2019.

T. Pranckevičius and V. Marcinkevičius. Application of logistic regression with part-of-the-speech tagging for multi-class text classification. In *2016 IEEE 4th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*, pages 1–5. IEEE, 2016.

A. Roy, K. Nikolitch, R. McGinn, S. Jinah, W. Klement, and Z. A. Kaminsky. A machine learning approach predicts future risk to suicidal ideation from social media data. *NPJ digital medicine*, 3(1):1–12, 2020.

S. Russell and P. Norvig. Artificial intelligence: a modern approach. 2002.

K. Shah, H. Patel, D. Sanghvi, and M. Shah. A comparative analysis of logistic regression, random forest and knn models for the text classification. *Augmented Human Research*, 5 (1):1–16, 2020.

J. H. Shen and F. Rudzicz. Detecting anxiety through reddit. In *Proceedings of the Fourth Workshop on Computational Linguistics and Clinical Psychology—From Linguistic Signal to Clinical Reality*, pages 58–65, 2017.

R. Socher, Y. Bengio, and C. D. Manning. Deep learning for nlp (without magic). In *Tutorial Abstracts of ACL 2012*, pages 5–5. 2012.

R. Stewart and S. Velupillai. Applied natural language processing in mental health big data. *Neuropsychopharmacology*, 46(1):252, 2021.

C. Strobl, J. Malley, and G. Tutz. An introduction to recursive partitioning: rationale, application, and characteristics of classification and regression trees, bagging, and random forests. *Psychological methods*, 14(4):323, 2009.

I. Strumberger, E. Tuba, N. Bacanin, R. Jovanovic, and M. Tuba. Convolutional neural network architecture design by the tree growth algorithm framework. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.

J. Su and H. Zhang. A fast decision tree learning algorithm. In *Aaai*, volume 6, pages 500–505, 2006.

S. Sudholt and G. A. Fink. Phocnet: A deep convolutional neural network for word spotting in handwritten documents. In *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 277–282. IEEE, 2016.

V. Svetnik, A. Liaw, C. Tong, J. C. Culberson, R. P. Sheridan, and B. P. Feuston. Random forest: a classification and regression tool for compound classification and qsar modeling. *Journal of chemical information and computer sciences*, 43(6):1947–1958, 2003.

C. Troussas, M. Virvou, K. J. Espinosa, K. Llaguno, and J. Caro. Sentiment analysis of facebook statuses using naive bayes classifier for language learning. In *IISA 2013*, pages 1–6. IEEE, 2013.

H. Van der Aa, J. Carmona Vargas, H. Leopold, J. Mendling, and L. Padró. Challenges and opportunities of applying natural language processing in business process management. In *COLING 2018: The 27th International Conference on Computational Linguistics: Proceedings of the Conference: August 20-26, 2018 Santa Fe, New Mexico, USA*, pages 2791–2801. Association for Computational Linguistics, 2018.

S. Wshah, C. Skalka, M. Price, et al. Predicting posttraumatic stress disorder risk: a machine learning approach. *JMIR mental health*, 6(7):e13946, 2019.

B. Xu, X. Guo, Y. Ye, and J. Cheng. An improved random forest classifier for text categorization. *J. Comput.*, 7(12):2913–2920, 2012.

S. Xu, Y. Li, and Z. Wang. Bayesian multinomial naïve bayes classifier to text classification. In *Advanced multimedia and ubiquitous engineering*, pages 347–352. Springer, 2017.

Z. Yong, L. Youwen, and X. Shixiong. An improved knn text classification algorithm based on clustering. *Journal of computers*, 4(3):230–237, 2009.

Appendix A

Supplementary Figures

A.1 Psychological Construct Neural Network Loss, Accuracy, and ROC curves.

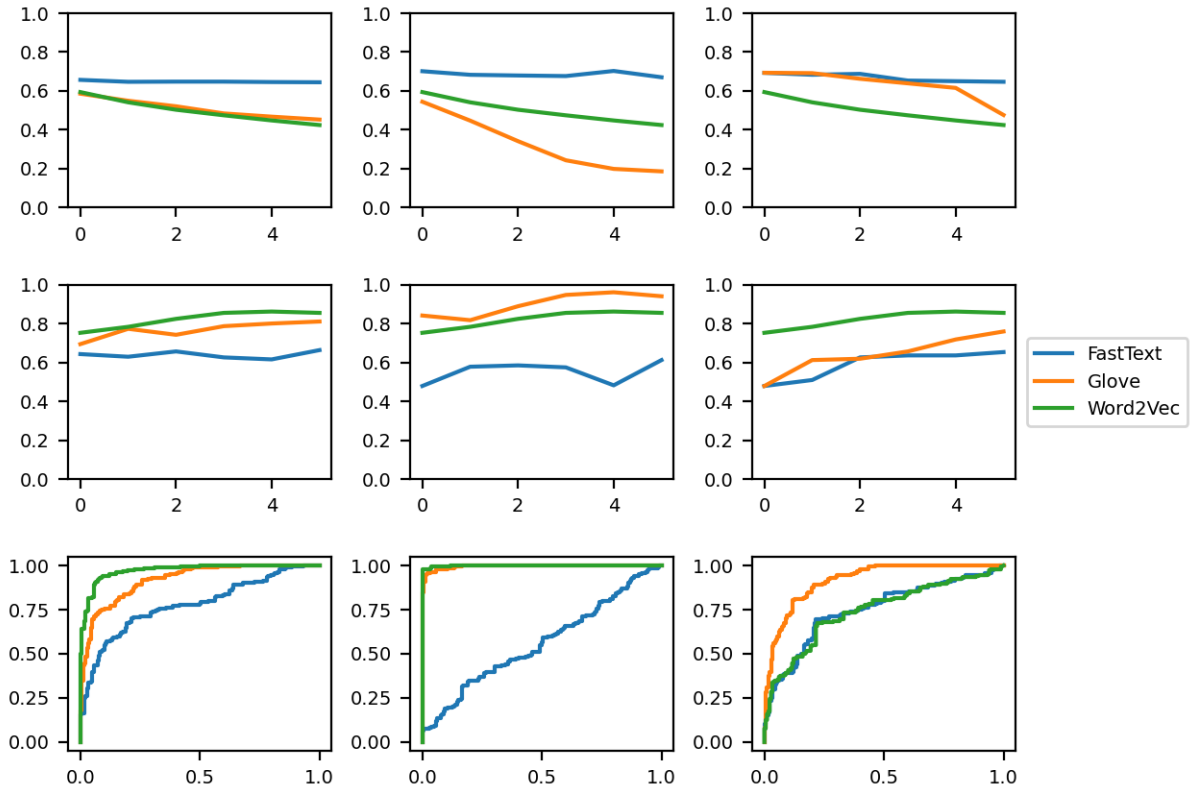

Figure A.1: *Three neural network and word embedding performance for the psychological construct burden. Three metrics loss* `(top row)`*, accuracy* `(middle row)`*, and ROC* `(bottom row)` *for models SNN* `(left)`*, CNN* `(center)`*, and LSTM* `(right)`*. Each line represents the word embedding methods: FastText* `(blue)`*, Glove* `(yellow)`*, and Word2Vec* `(green)`*.*

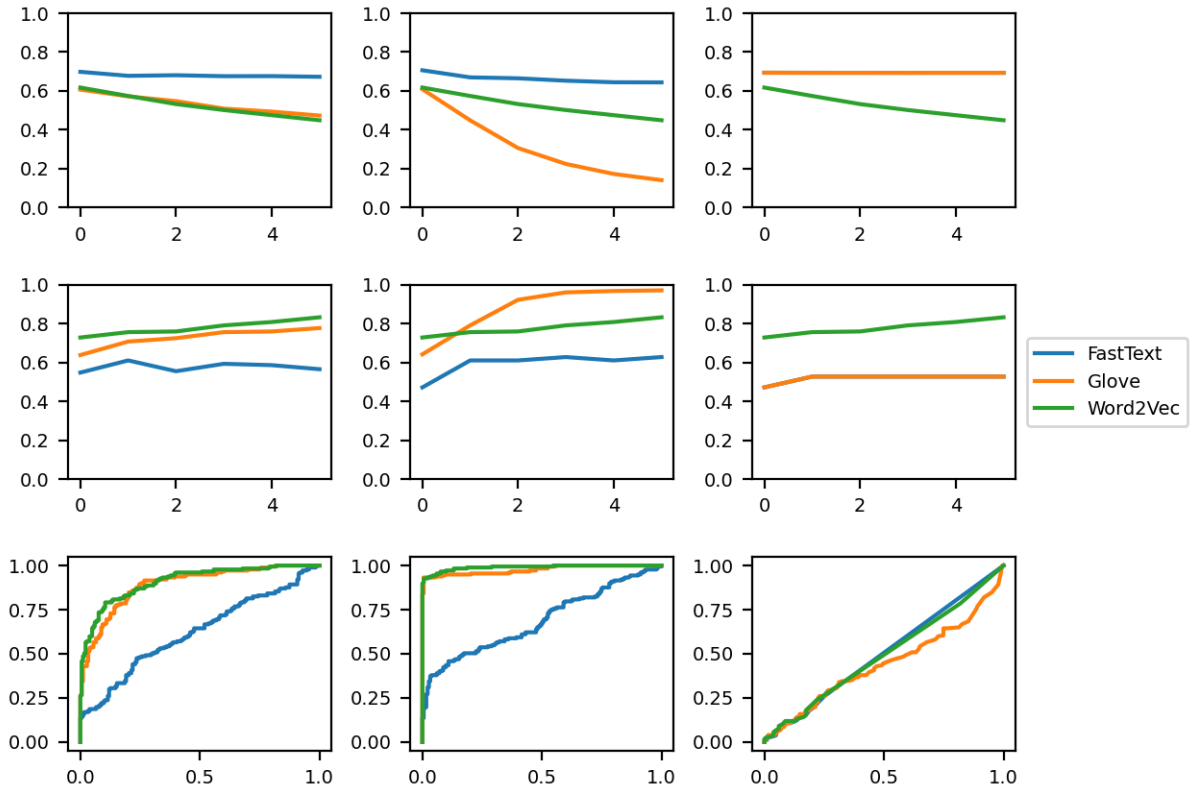

Figure A.2: *Three neural network and word embedding performance for the psychological construct depression 1. Three metrics loss* `(top row)`*, accuracy* `(middle row)`*, and ROC* `(bottom row)` *for models SNN* `(left)`*, CNN* `(center)`*, and LSTM* `(right)`*. Each line represents the word embedding methods: FastText* `(blue)`*, Glove* `(yellow)`*, and Word2Vec* `(green)`*.*

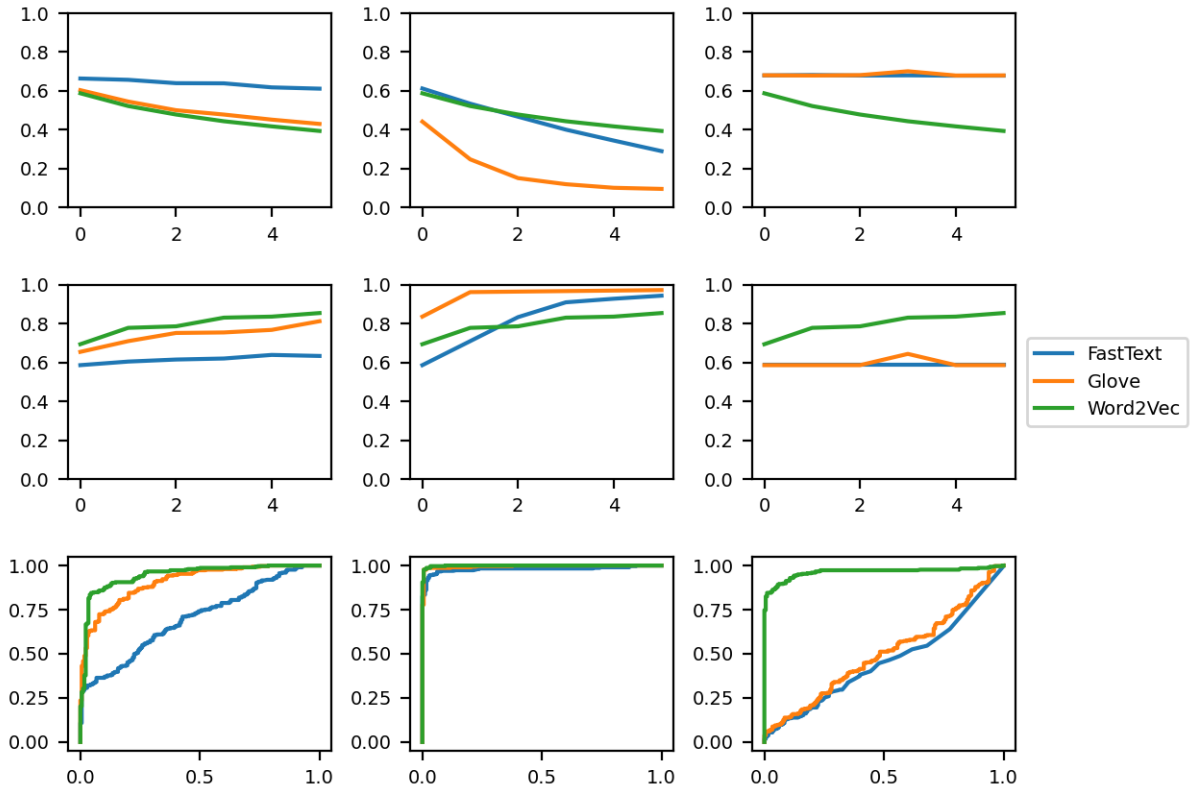

Figure A.3: *Three neural network and word embedding performance for the psychological construct depression 2. Three metrics loss* `(top row)`*, accuracy* `(middle row)`*, and ROC* `(bottom row)` *for models SNN* `(left)`*, CNN* `(center)`*, and LSTM* `(right)`*. Each line represents the word embedding methods: FastText* `(blue)`*, Glove* `(yellow)`*, and Word2Vec* `(green)`*.*

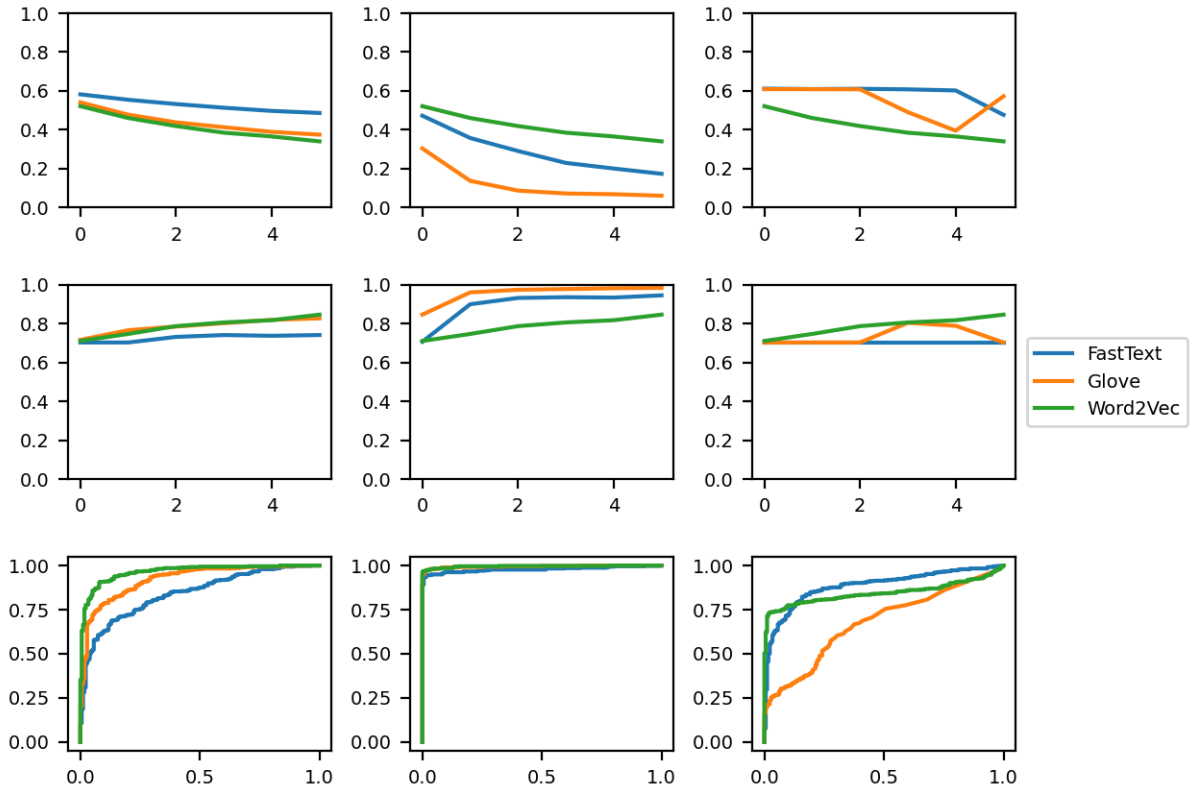

Figure A.4: *Three neural network and word embedding performance for the psychological construct depression 3. Three metrics loss* `(top row)`*, accuracy* `(middle row)`*, and ROC* `(bottom row)` *for models SNN* `(left)`*, CNN* `(center)`*, and LSTM* `(right)`*. Each line represents the word embedding methods: FastText* `(blue)`*, Glove* `(yellow)`*, and Word2Vec* `(green)`*.*

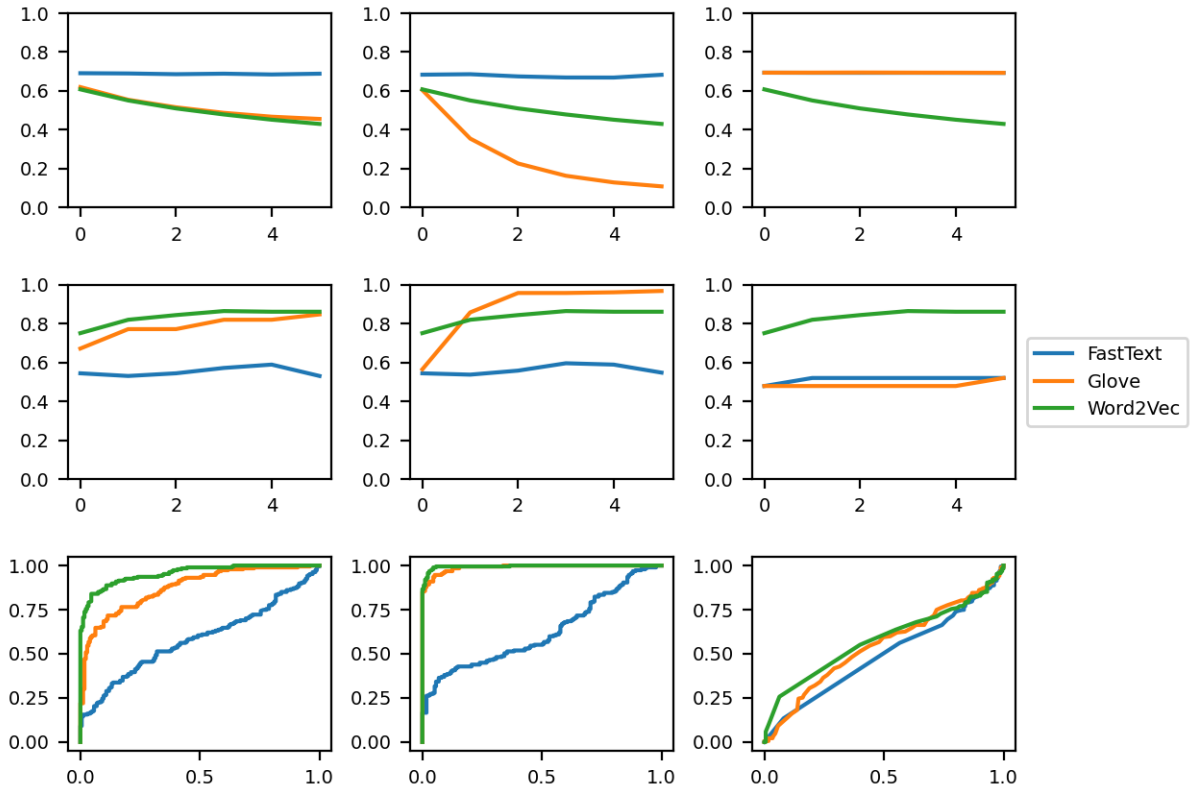

Figure A.5: *Three neural network and word embedding performance for the psychological construct hopelessness. Three metrics loss* `(top row)`*, accuracy* `(middle row)`*, and ROC* `(bottom row)` *for models SNN* `(left)`*, CNN* `(center)`*, and LSTM* `(right)`*. Each line represents the word embedding methods: FastText* `(blue)`*, Glove* `(yellow)`*, and Word2Vec* `(green)`*.*

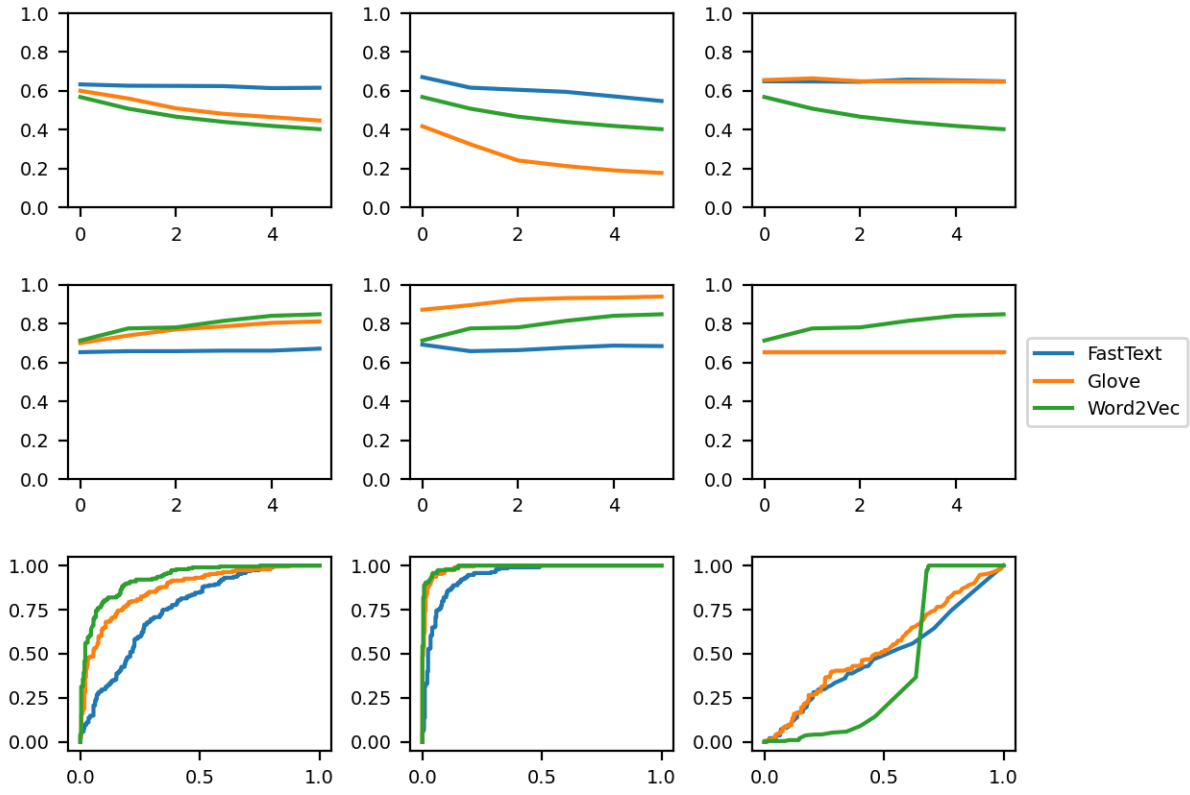

Figure A.6: *Three neural network and word embedding performance for the psychological construct insomnia. Three metrics loss* `(top row)`*, accuracy* `(middle row)`*, and ROC* `(bottom row)` *for models SNN* `(left)`*, CNN* `(center)`*, and LSTM* `(right)`*. Each line represents the word embedding methods: FastText* `(blue)`*, Glove* `(yellow)`*, and Word2Vec* `(green)`*.*

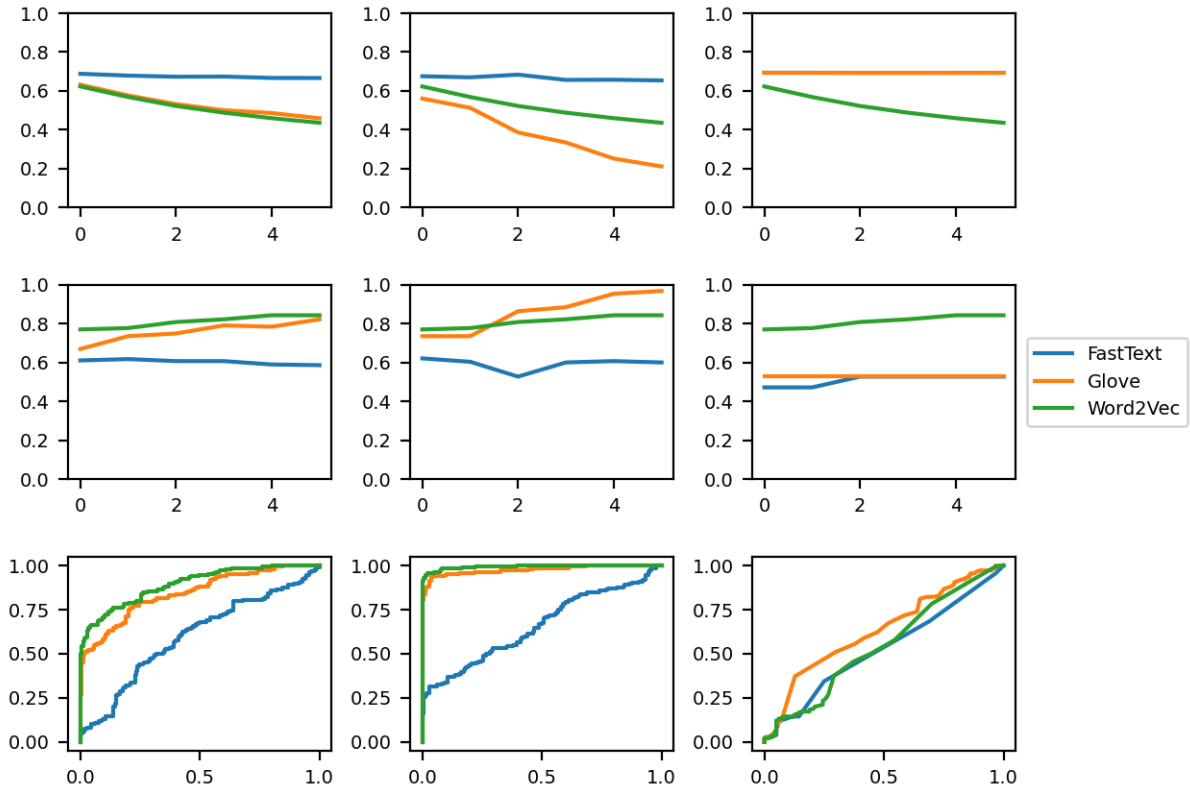

Figure A.7: *Three neural network and word embedding performance for the psychological construct loneliness. Three metrics loss* `(top row)`*, accuracy* `(middle row)`*, and ROC* `(bottom row)` *for models SNN* `(left)`*, CNN* `(center)`*, and LSTM* `(right)`*. Each line represents the word embedding methods: FastText* `(blue)`*, Glove* `(yellow)`*, and Word2Vec* `(green)`*.*

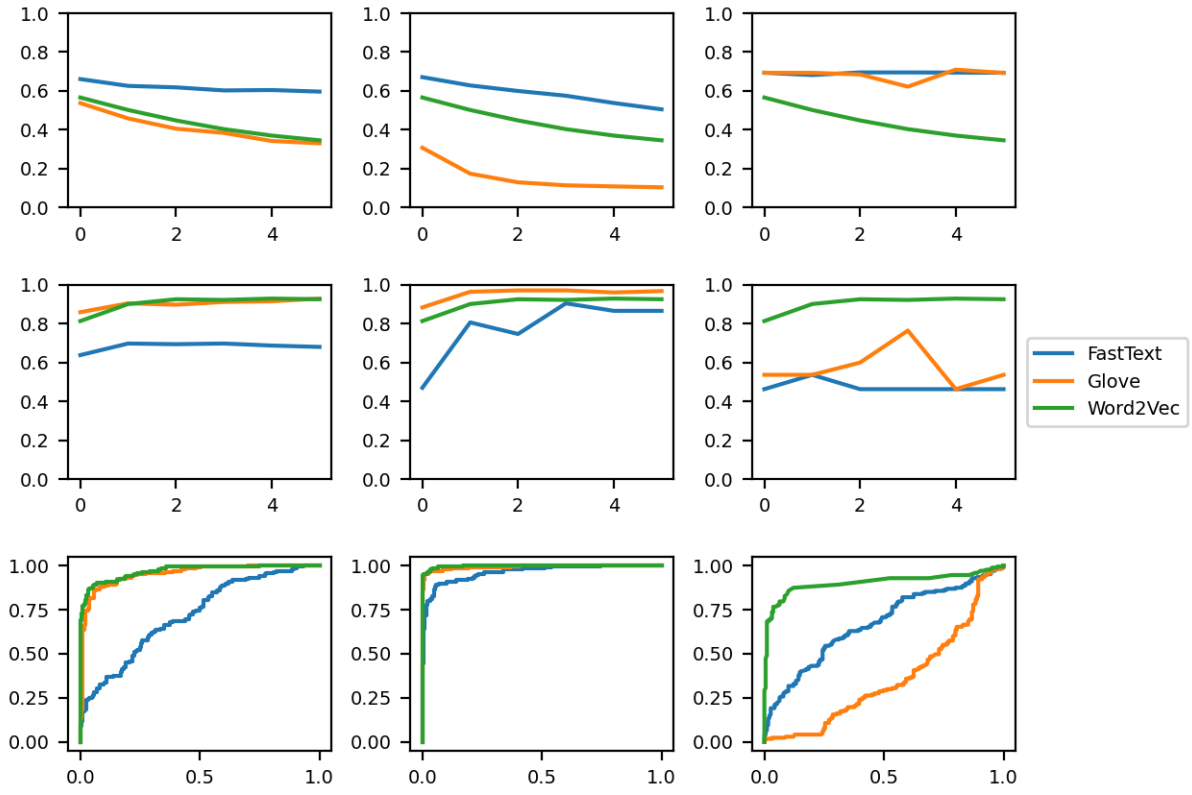

Figure A.8: *Three neural network and word embedding performance for the psychological construct stress. Three metrics loss* `(top row)`*, accuracy* `(middle row)`*, and ROC* `(bottom row)` *for models SNN* `(left)`*, CNN* `(center)`*, and LSTM* `(right)`*. Each line represents the word embedding methods: FastText* `(blue)`*, Glove* `(yellow)`*, and Word2Vec* `(green)`*.*

Appendix B

Supplementary Python Code

B.1 Extract Twitter SI

```
# For sending GET requests from the API
import requests
# For saving access tokens and for file management
# when creating and adding to thedataset
import os
# For dealing with json responses we receive from the API
import json
# For displaying the data after
import pandas as pd
# For saving the response data in CSV format
import csv
# For parsing the dates received from twitter in
# readable formats
import datetime
import dateutil.parser
import unicodedata
```

```
#To add wait time between requests
import time

def auth():
    return os.getenv('TOKEN')

def create_headers(bearer_token):
    headers = {"Authorization": "Bearer {}".format(bearer_token)}
    return headers

def create_url(keyword, start_date, end_date, max_results):

    search_url = "https://api.twitter.com/2/tweets/search/all"
    #Change to the endpoint you want to collect data from

    #change params based on the endpoint you are using
    query_params = {'query': keyword,
                    'start_time': start_date,
                    'end_time': end_date,
                    'max_results': max_results,
                    'expansions': 'author_id,in_reply_to_user_id,geo.place_id',
                    'tweet.fields': 'id,text,author_id,in_reply_to_user_id,
                    geo,conversation_id,created_at,lang,
                    public_metrics,referenced_tweets,reply_settings,source',
                    'user.fields': 'id,name,username,created_at,
                    description,public_metrics,verified',
                    'place.fields': 'full_name,id,country,country_code,
                    geo,name,place_type',
                    'next_token': {}}
```

```
    return (search_url, query_params)


def connect_to_endpoint(url, headers, params, next_token = None):
    params['next_token'] = next_token
    #params object received from create_url function
    response = requests.request("GET", url, headers = headers,
                                params = params)
    print("Endpoint Response Code: " + str(response.status_code))
    if response.status_code != 200:
        raise Exception(response.status_code, response.text)
    return response.json()


def append_to_csv(json_response, fileName):
    #A counter variable
    counter = 0
    #Open OR create the target CSV file
    csvFile = open(fileName, "a", newline="", encoding='utf-8')
    csvWriter = csv.writer(csvFile)
    #Loop through each tweet
    for tweet in json_response['data']:
        # create a variable for each since some of the keys
        # might not exist for some tweets so we will account for that
        # Author ID
        author_id = tweet['author_id']
        # Time created
        created_at = dateutil.parser.parse(tweet['created_at'])
        # Tweet ID
        tweet_id = tweet['id']
        # Language
```

```
        lang = tweet['lang']
        # Tweet metrics
        retweet_count = tweet['public_metrics']['retweet_count']
        reply_count = tweet['public_metrics']['reply_count']
        like_count = tweet['public_metrics']['like_count']
        quote_count = tweet['public_metrics']['quote_count']
        # source
        source = tweet['source']
        # Tweet text
        text = tweet['text']
        # Assemble all data in a list
        res = [author_id, created_at, tweet_id, lang, like_count,
               quote_count, reply_count, retweet_count, source, text]
        # Append the result to the CSV file
        csvWriter.writerow(res)
        counter += 1
    # When done, close the CSV file
    csvFile.close()
```

B.2 Extract SI and Control Timelines

```
# For sending GET requests from the API
import requests
# For saving access tokens and for file management when creating
# and adding to the dataset
import os
# For dealing with json responses we receive from the API
import json
# For displaying the data after
```

```
import pandas as pd
# For saving the response data in CSV format
import csv
# For parsing the dates received from twitter in readable formats
import datetime
import dateutil.parser
import unicodedata
#To add wait time between requests
import time
import datetime
from datetime import datetime
from dateutil.relativedelta import relativedelta


def append_to_csv(json_response1, fileName):

    #A counter variable
    counter = 0

    #Open OR create the target CSV file
    csvFile = open(fileName, "a", newline="", encoding='utf-8')
    csvWriter = csv.writer(csvFile)

    #Loop through each tweet
    for tweet in json_response1['data']:

        # We will create a variable for each since some of the keys
        # might not exist for some tweets so we will account for that
```

```
        # Time created
        created_at = dateutil.parser.parse(tweet['created_at'])

        # User ID
        user_id = tweet['author_id']

        # Tweet text
        text = tweet['text']

        # Assemble all data in a list
        res = [user_id, created_at, text]

        # Append the result to the CSV file
        csvWriter.writerow(res)
        counter += 1

    # When done, close the CSV file

def auth():
    return os.getenv('TOKEN')

def create_headers(bearer_token):
    headers = {"Authorization": "Bearer {}".format(bearer_token)}
    return headers

def create_url(user_id, end_date, max_results = 10):

    search_url = "https://api.twitter.com/2/users/{}/tweets".format(user_id)
    #Change to the endpoint you want to collect data from
```

```
    #change params based on the endpoint you are using
    query_params = {'end_time': end_date,
                    'max_results': max_results,
                    'tweet.fields': 'id,text,author_id,in_reply_to_user_id,
                                    geo,conversation_id,
                                    created_at,lang,public_metrics,
                                    referenced_tweets,
                                    reply_settings,source',
                    'next_token': {}}
    return (search_url, query_params)


def bearer_oauth(r):
        """
        Method required by bearer token authentication.
        """
        r.headers["Authorization"] = f"Bearer {bearer_token}"
        r.headers["User-Agent"] = "v2UserTweetsPython"
        return r


def connect_to_endpoint(url, headers, params, next_token = None):
    params['next_token'] = next_token
    #params object received from create_url function
    response = requests.request("GET", url, headers = headers, params = params)
    print("Endpoint Response Code: " + str(response.status_code))
    if response.status_code != 200:
        raise Exception(response.status_code, response.text)
```

```
    return response.json()


def main():
    url = create_url()
    params = get_params()
    json_response1 = connect_to_endpoint(url, params)
    print(json.dumps(json_response, indent=4, sort_keys=True))
    if __name__ == "__main__":
        main()


#Create token, headers, and extract each tweet Author ID and posting time
bearer_token = auth()
headers = create_headers(bearer_token)
df = pd.read_csv('data.csv')
SIlist = df['author_id'].to_list()
start_list = df['created_at'].to_list()
start_list2 = [str(i.replace(' ', 'T')) for i in start_list]
#Convert start times to correct format for Twitter API
start_list2 = [i[:-6] for i in start_list2]
end_list2 = [i + 'Z' for i in start_list2]
display(end_list2[:-500])


#Convert end times to correct format for Twitter
end_date2 = []
for i in range(len(end_list2)):
    a = end_list2[i]
    end_date2.append(a.replace(' ', 'T')[:-2])
display(end_date2[:5])
```

```
bearer_token = auth()
headers = create_headers(bearer_token)
max_results = 100
#Total number of tweets we collected from the loop
total_tweets = 0
# Create file
csvFile = open("timeline.csv", "a", newline="", encoding='utf-8')
csvWriter = csv.writer(csvFile)
#Create headers for the data you want to save, in this example,
# we only want save these columns in our dataset
csvWriter.writerow(['author_id', 'created_at', 'text'])
csvFile.close()
#Loop through SI IDs and collect user timelines
for i in range(len(SIlist)):
        SI = SIlist[i]
        end_date = end_list2[i]

    # Inputs
        count = 0 # Counting tweets per time period
        max_count = 75 # Max tweets per time period
        flag = True
        next_token = None
        try:
    # Check if flag is true
            while flag:
        # Check if max_count reached
                if count >= max_count:
                    break
                print("-------------------")
```

```
            print("Token: ", next_token)
            url = create_url(SI, end_date, max_results)
            json_response = connect_to_endpoint(url[0],
                                                headers, url[1],
                                                next_token = None)
            result_count = json_response['meta']['result_count']

            if 'next_token' in json_response['meta']:
        # Save the token to use for next call
                next_token = json_response['meta']['next_token']
                print("Next Token: ", next_token)
                if result_count is not None and result_count > 0 and next_token is n
                    print("User ID: ", SI)
                    append_to_csv(json_response, "timeline.csv")
                    count += result_count
                    total_tweets += result_count
                    print("Total # of Tweets added: ", total_tweets)
                    print("-------------------")
                    time.sleep(5)
    # If no next token exists
            else:
                if result_count is not None and result_count > 0:
                    print("-------------------")
                    print("End Date: ", end_date)
                    append_to_csv(json_response, "timeline.csv")
                    count += result_count
                    total_tweets += result_count
                    print("Total # of Tweets added: ", total_tweets)
                    print("-------------------")
```

```
                        time.sleep(5)

            #Since this is the final request, turn flag to
            # false to move to the next time period.
                flag = False
                next_token = None
            time.sleep(5)
        except:
            pass
print("Total number of results: ", total_tweets)
```

B.3 Collect Psychological Constructs and Controls

To collect all psychological construct and controls we use the queries from Table 3.2.

```
# For sending GET requests from the API
import requests
# For saving access tokens and for file management
# when creating and adding to the dataset
import os
# For dealing with json responses we receive from the API
import json
# For displaying the data after
import pandas as pd
# For saving the response data in CSV format
import csv
# For parsing the dates received from twitter in readable formats
import datetime
import dateutil.parser
```

```
import unicodedata
#To add wait time between requests
import time

def auth():
    return os.getenv('TOKEN')

def create_headers(bearer_token):
    headers = {"Authorization": "Bearer {}".format(bearer_token)}
    return headers

def create_url(keyword, start_date, end_date, max_results):

    search_url = "https://api.twitter.com/2/tweets/search/all"
    #Change to the endpoint you want to collect data from

    #change params based on the endpoint you are using
    query_params = {'query': keyword,
                    'start_time': start_date,
                    'end_time': end_date,
                    'max_results': max_results,
                    'expansions': 'author_id,in_reply_to_user_id,geo.place_id',
                    'tweet.fields': 'id,text,author_id,in_reply_to_user_id,
                                     geo,conversation_id,created_at,lang,
                                     public_metrics,referenced_tweets,
                                     reply_settings,source',
                    'user.fields': 'id,name,username,created_at,
                                     description,public_metrics,verified',
                    'place.fields': 'full_name,id,country,
```

```
                                    country_code,geo,
                                    name,place_type',
                    'next_token': {}}
    return (search_url, query_params)

def connect_to_endpoint(url, headers, params, next_token = None):
    params['next_token'] = next_token
    #params object received from create_url function
    response = requests.request("GET", url, headers = headers, params = params)
    print("Endpoint Response Code: " + str(response.status_code))
    if response.status_code != 200:
        raise Exception(response.status_code, response.text)
    return response.json()

def append_to_csv(json_response, fileName):
    #A counter variable
    counter = 0
    #Open OR create the target CSV file
    csvFile = open(fileName, "a", newline="", encoding='utf-8')
    csvWriter = csv.writer(csvFile)
    #Loop through each tweet
    for tweet in json_response['data']:
        # create a variable for each since some of the keys might
        # not exist for some tweets so we will account for that
        # Author ID
        author_id = tweet['author_id']
        # Time created
        created_at = dateutil.parser.parse(tweet['created_at'])
        # Tweet ID
```

```
        tweet_id = tweet['id']
        # Language
        lang = tweet['lang']
        # Tweet metrics
        retweet_count = tweet['public_metrics']['retweet_count']
        reply_count = tweet['public_metrics']['reply_count']
        like_count = tweet['public_metrics']['like_count']
        quote_count = tweet['public_metrics']['quote_count']
        # source
        source = tweet['source']
        # Tweet text
        text = tweet['text']
        # Assemble all data in a list
        res = [author_id, created_at, tweet_id,
               lang, like_count, quote_count, reply_count,
               retweet_count, source, text]
        # Append the result to the CSV file
        csvWriter.writerow(res)
        counter += 1
    # When done, close the CSV file
    csvFile.close()

start_list = ['2019-01-01T00:00:00.000Z',
              '2019-02-01T00:00:00.000Z',
              '2019-03-01T00:00:00.000Z',
              '2019-04-01T00:00:00.000Z',
              '2019-05-01T00:00:00.000Z',
              '2019-06-01T00:00:00.000Z',
              '2019-07-01T00:00:00.000Z',
```

```
                '2019-08-01T00:00:00.000Z',
                '2019-09-01T00:00:00.000Z',
                '2019-10-01T00:00:00.000Z',
                '2019-11-01T00:00:00.000Z',
                '2019-12-01T00:00:00.000Z',
                '2020-01-01T00:00:00.000Z',
                '2020-02-01T00:00:00.000Z',
                '2020-03-01T00:00:00.000Z',
                '2020-04-01T00:00:00.000Z',
                '2020-05-01T00:00:00.000Z',
                '2020-06-01T00:00:00.000Z',
                '2020-07-01T00:00:00.000Z',
                '2020-08-01T00:00:00.000Z',
                '2020-09-01T00:00:00.000Z',
                '2020-10-01T00:00:00.000Z',
                '2020-11-01T00:00:00.000Z',
                '2020-12-01T00:00:00.000Z',
                '2021-01-01T00:00:00.000Z',
                '2021-02-01T00:00:00.000Z',
                '2021-03-01T00:00:00.000Z',
                '2021-04-01T00:00:00.000Z',
                '2021-05-01T00:00:00.000Z',
                '2021-06-01T00:00:00.000Z',
                '2021-07-01T00:00:00.000Z',
                '2021-08-01T00:00:00.000Z',]

end_list =  ['2019-01-31T00:00:00.000Z',
             '2019-02-28T00:00:00.000Z',
             '2019-03-31T00:00:00.000Z',
```

```
'2019-04-30T00:00:00.000Z',
'2019-05-31T00:00:00.000Z',
'2019-06-30T00:00:00.000Z',
'2019-07-31T00:00:00.000Z',
'2019-08-31T00:00:00.000Z',
'2019-09-30T00:00:00.000Z',
'2019-10-31T00:00:00.000Z',
'2019-11-30T00:00:00.000Z',
'2019-12-31T00:00:00.000Z',
'2020-01-31T00:00:00.000Z',
'2020-02-29T00:00:00.000Z',
'2020-03-31T00:00:00.000Z',
'2020-04-30T00:00:00.000Z',
'2020-05-31T00:00:00.000Z',
'2020-06-30T00:00:00.000Z',
'2020-07-31T00:00:00.000Z',
'2020-08-31T00:00:00.000Z',
'2020-09-30T00:00:00.000Z',
'2020-10-31T00:00:00.000Z',
'2020-11-30T00:00:00.000Z',
'2020-12-31T00:00:00.000Z',
'2021-01-31T00:00:00.000Z',
'2021-02-28T00:00:00.000Z',
'2021-03-31T00:00:00.000Z',
'2021-04-30T00:00:00.000Z',
'2021-05-31T00:00:00.000Z',
'2021-06-30T00:00:00.000Z',
'2021-07-31T00:00:00.000Z',
'2021-08-31T00:00:00.000Z',]
```

```
bearer_token = auth()
headers = create_headers(bearer_token)
keyword = #Insert desired query as string format here#

max_results = 30

#Total number of tweets we collected from the loop
total_tweets = 0

# Create file
csvFile = open("anxiety.csv", "a", newline="", encoding='utf-8')
csvWriter = csv.writer(csvFile)

#Create headers for the data you want to save,
#in this example, we only want save these columns in our dataset
csvWriter.writerow(['author_id', 'created_at',
                    'id','lang', 'like_count', 'quote_count', 'reply_count','retweet_cou
csvFile.close()

for i in range(0,len(start_list)):
    # Inputs
    count = 0 # Counting tweets per time period
    max_count = 10 # Max tweets per time period
    flag = True
    next_token = None

    # Check if flag is true
    while flag:
```

```
# Check if max_count reached
if count >= max_count:
    break
print("-------------------")
print("Token: ", next_token)
url = create_url(keyword, start_list[i],end_list[i], max_results)
json_response = connect_to_endpoint(url[0], headers, url[1], next_token)
result_count = json_response['meta']['result_count']

if 'next_token' in json_response['meta']:
    # Save the token to use for next call
    next_token = json_response['meta']['next_token']
    print("Next Token: ", next_token)
    if result_count is not None and result_count > 0 and next_token is not None:
        print("Start Date: ", start_list[i])
        append_to_csv(json_response, "anxiety.csv")
        count += result_count
        total_tweets += result_count
        print("Total # of Tweets added: ", total_tweets)
        print("-------------------")
        time.sleep(5)
# If no next token exists
else:
    if result_count is not None and result_count > 0:
        print("-------------------")
        print("Start Date: ", start_list[i])
        append_to_csv(json_response, "anxiety.csv")
        count += result_count
        total_tweets += result_count
```

```
                print("Total # of Tweets added: ", total_tweets)
                print("--------------------")
                time.sleep(5)

            # Since this is the final request,
            # turn flag to false to move to the next time period.
            flag = False
            next_token = None
        time.sleep(5)
print("Total number of results: ", total_tweets)
```

B.4 Neural Network Evaluation

To evaluate the best type of neural network and pre-trained word embedding to use for psychological construct we use the following code:

B.4.1 Word2Vec

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import re
import nltk
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
from gensim.models import FastText
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from numpy import array
```

```
from keras.preprocessing.text import one_hot
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers.core import Activation, Dropout, Dense
from keras.layers import Flatten, Conv1D, LSTM
from keras.layers import GlobalMaxPool1D
from keras.layers.embeddings import Embedding
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer

anxiety = pd.read_csv('/Users/tylerbastian/Thesis/Psych Constructs/anxiety.csv')
control = pd.read_csv('/Users/tylerbastian/Thesis/Psych Constructs/anxietyControl.csv')
anxious_tweets = anxiety['tweet']
control_tweets = control['tweet']

def remove_emoji(string):
    emoji_pattern = re.compile("["
                               u"\U0001F600-\U0001F64F"  # emoticons
                               u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                               u"\U0001F680-\U0001F6FF"  # transport & map symbols
                               u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                               u"\U00002500-\U00002BEF"  # chinese char
                               u"\U00002702-\U000027B0"
                               u"\U00002702-\U000027B0"
                               u"\U000024C2-\U0001F251"
                               u"\U0001f926-\U0001f937"
                               u"\U00010000-\U0010ffff"
                               u"\u2640-\u2642"
                               u"\u2600-\u2B55"
```

```
                                    u"\u200d"
                                    u"\u23cf"
                                    u"\u23e9"
                                    u"\u231a"
                                    u"\ufe0f"  # dingbats
                                    u"\u3030"
                                    "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', string)

#Case tweet cleaning
text = []
for i in anxious_tweets:
    text.append(re.sub(r"(?:\@|https?\://)\S+", "", i))
text2 = []
for i in text:
    text2.append(re.sub(r"@\w+", "", i).replace("RT",''))
text3 = []
for i in text2:
    text3.append(re.sub(r"#\w+","",i))
text4 = []
for i in text3:
    text4.append(remove_emoji(i))
tokenizer = RegexpTokenizer(r'[a-zA-Z]+')
tweet_token = []
for x in text4:
    tweet_token.append(tokenizer.tokenize(x.lower()))
anxious_lengths = []
for i in range(len(tweet_token)):
    anxious_lengths.append(len(tweet_token[i]))
```

```
#Control tweet cleaning
text = []
for i in control_tweets:
    text.append(re.sub(r"(?:\@|https?\://)\S+", "", i))
text2 = []
for i in text:
    text2.append(re.sub(r"@\w+", "", i).replace("RT",''))
text3 = []
for i in text2:
    text3.append(re.sub(r"#\w+","",i))
text4 = []
for i in text3:
    text4.append(remove_emoji(i))
tokenizer = RegexpTokenizer(r'[a-zA-Z]+')
control_token = []
for x in text4:
    control_token.append(tokenizer.tokenize(x.lower()))
control_lengths = []
for i in range(len(control_token)):
    control_lengths.append(len(control_token[i]))

X = []
for words in control_token:
    X.append(' '.join(words))
Y = []
for words in tweet_token:
    Y.append(' '.join(words))
```

```
controls = pd.DataFrame(X)
controls.insert(0,'Sentiment','negative')
anxious = pd.DataFrame(Y)
anxious.insert(0,'Sentiment','positive')

#Graph proportionality of sentiments

tweets = pd.concat([anxious,controls])
X = tweets[0]
y = tweets['Sentiment']
y = np.array(list(map(lambda x: 1 if x=="positive" else 0, y)))
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=42)

tokenizer = Tokenizer(num_words = 5000)
tokenizer.fit_on_texts(X_train)
X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)

vocab_size = len(tokenizer.word_index)+1
maxlen = 100
X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)

### Load Word2Vec embeddings
from numpy import array
from numpy import asarray
from numpy import zeros
```

```
bin_file = '/Users/tylerbastian/Desktop/Spring 2022
            /Thesis/Embeddings/GoogleNews-vectors-negative300.bin'
word2vec = KeyedVectors.load_word2vec_format(bin_file, binary=True,
                                             unicode_errors='ignore')
non_exist = []
embedding_matrix = zeros((vocab_size, 300))
for word, index in tokenizer.word_index.items():
    if word not in word2vec.index_to_key:
        non_exist.append(word)
    else:
        embedding_vector = word2vec.get_vector((word))
        if embedding_vector is not None:
            embedding_matrix[index] = embedding_vector
#Text Classification with Neural Network
model = Sequential()
embedding_layer = Embedding(vocab_size, 300, weights=[embedding_matrix],
                            input_length=maxlen, trainable=False)
model.add(embedding_layer)
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))

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

history = model.fit(X_train, y_train, batch_size = 128,
                    epochs = 6, verbose = 1, validation_split = 0.2)

score = model.evaluate(X_test, y_test, verbose = 1)
```

```
print("Test Score:", score[0])
print("Test Accuracy:", score[1])

plt.subplot(211)
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.subplot(212)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.tight_layout()
plt.savefig('Word2Vec_SNN.png')

#Text classification with CNN
model=Sequential()

embedding_layer = Embedding(vocab_size, 300,
                            weights=[embedding_matrix],
                            input_length=maxlen,
                            trainable=False)
model.add(embedding_layer)
model.add(Conv1D(128, 5, activation='relu'))
```

```
model.add(GlobalMaxPool1D())
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy',
              metrics=['acc'])
print(model.summary())

history = model.fit(X_train, y_train, batch_size=128,
                    epochs = 6, verbose=1, validation_split=0.2)
score = model.evaluate(X_test, y_test, verbose=1)
print("Test Score:", score[0])
print("Test Accuracy:", score[1])

plt.subplot(211)
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.subplot(212)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.tight_layout()
plt.savefig('Word2Vec_CNN.png')
```

```
###LSTM Recurrent Neural Network
model = Sequential()
embedding_layer = Embedding(vocab_size, 300, weights=[embedding_matrix],
                            input_length=maxlen, trainable=False)
model.add(embedding_layer)
model.add(LSTM(128))

model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy',
              metrics=['acc'])
print(model.summary())

history = model.fit(X_train, y_train, batch_size=128,
                    epochs=6, verbose=1, validation_split=0.2)
score = model.evaluate(X_test, y_test, verbose=1)
print("Test Score:", score[0])
print("Test Accuracy:", score[1])

plt.subplot(211)
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.subplot(212)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
```

```
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.tight_layout()
plt.savefig('Word2Vec_LSTM.png')
```

B.4.2 Glove

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import re
import nltk
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from numpy import array
from keras.preprocessing.text import one_hot
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers.core import Activation, Dropout, Dense
from keras.layers import Flatten, Conv1D, LSTM
from keras.layers import GlobalMaxPool1D
from keras.layers.embeddings import Embedding
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
from sklearn.metrics import roc_curve, auc

anxiety = pd.read_csv('/Users/tylerbastian/Thesis/Psych Constructs/anxiety.csv')
control = pd.read_csv('/Users/tylerbastian/Thesis/Psych Constructs/anxietyControl.csv')
```

```
anxious_tweets = anxiety['tweet']
control_tweets = control['tweet']

def remove_emoji(string):
    emoji_pattern = re.compile("["
                               u"\U0001F600-\U0001F64F"  # emoticons
                               u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                               u"\U0001F680-\U0001F6FF"  # transport & map symbols
                               u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                               u"\U00002500-\U00002BEF"  # chinese char
                               u"\U00002702-\U000027B0"
                               u"\U00002702-\U000027B0"
                               u"\U000024C2-\U0001F251"
                               u"\U0001f926-\U0001f937"
                               u"\U00010000-\U0010ffff"
                               u"\u2640-\u2642"
                               u"\u2600-\u2B55"
                               u"\u200d"
                               u"\u23cf"
                               u"\u23e9"
                               u"\u231a"
                               u"\ufe0f"  # dingbats
                               u"\u3030"
                               "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', string)

#Case tweet cleaning
text = []
for i in anxious_tweets:
```

```
    text.append(re.sub(r"(?:\@|https?\://)\S+", "", i))
text2 = []
for i in text:
    text2.append(re.sub(r"@\w+", "", i).replace("RT",''))
text3 = []
for i in text2:
    text3.append(re.sub(r"#\w+","",i))
text4 = []
for i in text3:
    text4.append(remove_emoji(i))
tokenizer = RegexpTokenizer(r'[a-zA-Z]+')
tweet_token = []
for x in text4:
    tweet_token.append(tokenizer.tokenize(x.lower()))
anxious_lengths = []
for i in range(len(tweet_token)):
    anxious_lengths.append(len(tweet_token[i]))

#Control tweet cleaning
text = []
for i in control_tweets:
    text.append(re.sub(r"(?:\@|https?\://)\S+", "", i))
text2 = []
for i in text:
    text2.append(re.sub(r"@\w+", "", i).replace("RT",''))
text3 = []
for i in text2:
    text3.append(re.sub(r"#\w+","",i))
text4 = []
```

```
for i in text3:
    text4.append(remove_emoji(i))
tokenizer = RegexpTokenizer(r'[a-zA-Z]+')
control_token = []
for x in text4:
    control_token.append(tokenizer.tokenize(x.lower()))
control_lengths = []
for i in range(len(control_token)):
    control_lengths.append(len(control_token[i]))

X = []
for words in control_token:
    X.append(' '.join(words))
Y = []
for words in tweet_token:
    Y.append(' '.join(words))

controls = pd.DataFrame(X)
controls.insert(0,'Sentiment','negative')
anxious = pd.DataFrame(Y)
anxious.insert(0,'Sentiment','positive')

#Graph proportionality of sentiments
tweets = pd.concat([anxious,controls])
import seaborn as sns
sns.countplot(x='Sentiment', data=tweets)
#Relatively equal proportion of control and case

X = tweets[0]
```

```
y = tweets['Sentiment']
y = np.array(list(map(lambda x: 1 if x=="positive" else 0, y)))
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=42)
tokenizer = Tokenizer(num_words = 5000)
tokenizer.fit_on_texts(X_train)
X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)
vocab_size = len(tokenizer.word_index)+1
maxlen = 100
X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)

### Load Glove embeddings
from numpy import array
from numpy import asarray
from numpy import zeros
embeddings_dictionary = dict()
path = '/Users/tylerbastian/Desktop/Spring 2022/Thesis/Embeddings/glove.6B.100d.txt'
glove_file = open(path, encoding="utf8")

for line in glove_file:
    records = line.split()
    word = records[0]
    vector_dimensions = asarray(records[1:], dtype='float32')
    embeddings_dictionary [word] = vector_dimensions
glove_file.close()
```

```
embedding_matrix = zeros((vocab_size, 100))
for word, index in tokenizer.word_index.items():
    embedding_vector = embeddings_dictionary.get(word)
    if embedding_vector is not None:
        embedding_matrix[index] = embedding_vector

#Text Classification with Neural Network
model = Sequential()
embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix],
                            input_length=maxlen, trainable=False)
model.add(embedding_layer)
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))

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

history = model.fit(X_train, y_train, batch_size = 128,
                    epochs = 6, verbose = 1, validation_split = 0.2)
score = model.evaluate(X_test, y_test, verbose = 1)
print("Test Score:", score[0])
print("Test Accuracy:", score[1])

plt.subplot(211)
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
```

```
plt.xlabel('Epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.subplot(212)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.tight_layout()
plt.savefig('HopelessnessGlove_SNN.png')

#Text classification with CNN
model=Sequential()
embedding_layer = Embedding(vocab_size, 100,
                            weights=[embedding_matrix],
                            input_length=maxlen,
                            trainable=False)
model.add(embedding_layer)
model.add(Conv1D(128, 5, activation='relu'))
model.add(GlobalMaxPool1D())
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy',
              metrics=['acc'])
print(model.summary())

history = model.fit(X_train, y_train, batch_size=128,
                    epochs = 6, verbose=1, validation_split=0.2)
score = model.evaluate(X_test, y_test, verbose=1)
```

```
print("Test Score:", score[0])
print("Test Accuracy:", score[1])
y_pred = model.predict(X_test).ravel()
nn_fpr_keras, nn_tpr_keras, nn_thresholds_keras = roc_curve(y_test, y_pred)
plt.plot(nn_fpr_keras, nn_tpr_keras, marker='.',
         label='Neural Network (auc = %0.3f)' % auc_keras)

plt.subplot(211)
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.subplot(212)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.tight_layout()
plt.savefig('HopelessnessGlove_CNN.png')


###LSTM Recurrent Neural Network
model = Sequential()
embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix],
                            input_length=maxlen, trainable=False)
```

```
model.add(embedding_layer)
model.add(LSTM(128))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy',
              metrics=['acc'])
print(model.summary())
history = model.fit(X_train, y_train, batch_size=128,
                    epochs=6, verbose=1, validation_split=0.2)
score = model.evaluate(X_test, y_test, verbose=1)
print("Test Score:", score[0])
print("Test Accuracy:", score[1])

plt.subplot(211)
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.subplot(212)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.tight_layout()
plt.savefig('HopelessnessGlove_LSTM.png')
```

B.4.3 FastText

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import re
import nltk
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
from gensim.models import FastText
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from numpy import array
from keras.preprocessing.text import one_hot
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers.core import Activation, Dropout, Dense
from keras.layers import Flatten, Conv1D, LSTM
from keras.layers import GlobalMaxPool1D
from keras.layers.embeddings import Embedding
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer

construct = pd.read_csv('/Users/tylerbastian/Thesis/Psych Constructs/anxiety.csv')
control = pd.read_csv('/Users/tylerbastian/Thesis/Psych Constructs/anxietyControl.csv')
construct_tweets = construct['tweet']
control_tweets = control['tweet']

def remove_emoji(string):
```

```
    emoji_pattern = re.compile("["
                               u"\U0001F600-\U0001F64F"  # emoticons
                               u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                               u"\U0001F680-\U0001F6FF"  # transport & map symbols
                               u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                               u"\U00002500-\U00002BEF"  # chinese char
                               u"\U00002702-\U000027B0"
                               u"\U00002702-\U000027B0"
                               u"\U000024C2-\U0001F251"
                               u"\U0001f926-\U0001f937"
                               u"\U00010000-\U0010ffff"
                               u"\u2640-\u2642"
                               u"\u2600-\u2B55"
                               u"\u200d"
                               u"\u23cf"
                               u"\u23e9"
                               u"\u231a"
                               u"\ufe0f"  # dingbats
                               u"\u3030"
                               "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', string)

#Case tweet cleaning
text = []
for i in construct_tweets:
    text.append(re.sub(r"(?:\@|https?\://)\S+", "", i))
text2 = []
for i in text:
    text2.append(re.sub(r"@\w+", "", i).replace("RT",''))
```

```
text3 = []
for i in text2:
    text3.append(re.sub(r"#\w+","",i))
text4 = []
for i in text3:
    text4.append(remove_emoji(i))
tokenizer = RegexpTokenizer(r'[a-zA-Z]+')
tweet_token = []
for x in text4:
    tweet_token.append(tokenizer.tokenize(x.lower()))
construct_lengths = []
for i in range(len(tweet_token)):
    construct_lengths.append(len(tweet_token[i]))

#Control tweet cleaning
text = []
for i in control_tweets:
    text.append(re.sub(r"(?:\@|https?\://)\S+", "", i))
text2 = []
for i in text:
    text2.append(re.sub(r"@\w+", "", i).replace("RT",''))
text3 = []
for i in text2:
    text3.append(re.sub(r"#\w+","",i))
text4 = []
for i in text3:
    text4.append(remove_emoji(i))
tokenizer = RegexpTokenizer(r'[a-zA-Z]+')
control_token = []
```

```
for x in text4:
    control_token.append(tokenizer.tokenize(x.lower()))
control_lengths = []
for i in range(len(control_token)):
    control_lengths.append(len(control_token[i]))

X = []
for words in control_token:
    X.append(' '.join(words))
Y = []
for words in tweet_token:
    Y.append(' '.join(words))

controls = pd.DataFrame(X)
controls.insert(0,'Sentiment','negative')
construct = pd.DataFrame(Y)
construct.insert(0,'Sentiment','positive')

#Graph proportionality of sentiments
tweets = pd.concat([construct,controls])
X = tweets[0]
y = tweets['Sentiment']
y = np.array(list(map(lambda x: 1 if x=="positive" else 0, y)))
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=42)

tokenizer = Tokenizer(num_words = 5000)
tokenizer.fit_on_texts(X_train)
```

```
X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)

vocab_size = len(tokenizer.word_index)+1
maxlen = 100
X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)

### Load Word2Vec embeddings
from numpy import array
from numpy import asarray
from numpy import zeros
training_tweets = ''.join(['']).join(sub) for sub in tweets[0]])
with open('trainingtweets.txt', 'w') as f:
    f.write(training_tweets)

import fasttext
filen = '/Users/tylerbastian/PycharmProjects/Stat766/trainingtweets.txt'
ft = fasttext.train_unsupervised(filen, model='skipgram')
ft.save_model("hopelessnesstweets.bin")

embedding_matrix = zeros((vocab_size, 100))
for word, index in tokenizer.word_index.items():
    embedding_vector = ft.get_word_vector(word)
    if embedding_vector is not None:
        embedding_matrix[index] = embedding_vector

#Text Classification with Neural Network
model = Sequential()
```

```
embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix],
                            input_length=maxlen, trainable=False)
model.add(embedding_layer)
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))

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

history = model.fit(X_train, y_train, batch_size = 128,
                    epochs = 6, verbose = 1, validation_split = 0.2)

score = model.evaluate(X_test, y_test, verbose = 1)
print("Test Score:", score[0])
print("Test Accuracy:", score[1])

plt.subplot(211)
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.subplot(212)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
```

```
plt.xlabel('Epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.tight_layout()
plt.savefig('FastText_SNN.png')

#Text classification with CNN
model=Sequential()

embedding_layer = Embedding(vocab_size, 100,
                            weights=[embedding_matrix],
                            input_length=maxlen,
                            trainable=False)
model.add(embedding_layer)
model.add(Conv1D(128, 5, activation='relu'))
model.add(GlobalMaxPool1D())
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy',
              metrics=['acc'])
print(model.summary())

history = model.fit(X_train, y_train, batch_size=128,
                    epochs = 6, verbose=1, validation_split=0.2)
score = model.evaluate(X_test, y_test, verbose=1)
print("Test Score:", score[0])
print("Test Accuracy:", score[1])

plt.subplot(211)
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
```

```
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.subplot(212)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.tight_layout()
plt.savefig('FastText_CNN.png')

###LSTM Recurrent Neural Network
model = Sequential()
embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix],
                            input_length=maxlen, trainable=False)
model.add(embedding_layer)
model.add(LSTM(128))

model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy',
              metrics=['acc'])
print(model.summary())

history = model.fit(X_train, y_train, batch_size=128,
                    epochs=6, verbose=1, validation_split=0.2)
score = model.evaluate(X_test, y_test, verbose=1)
```

```
print("Test Score:", score[0])
print("Test Accuracy:", score[1])


plt.subplot(211)
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.subplot(212)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.tight_layout()
plt.savefig('FastText_LSTM.png')
```

B.5 Train and Evaluate Psychological Constructs

To train and evaluate psychological constructs as discussed in Chapter 4, we use the code below. We train and evaluate using the data collected using code in Appendix A Section Collect Psychological Constructs and Controls.

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

```
import nltk
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from numpy import array
from keras.models import load_model
from keras.preprocessing.text import one_hot
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers.core import Activation, Dropout, Dense
from keras.layers import Flatten, Conv1D, LSTM
from keras.layers import GlobalMaxPool1D
from keras.layers.embeddings import Embedding
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
import keras

case = pd.read_csv('/Users/tylerbastian/Thesis/Psych Constructs/hopelessness.csv')
control = pd.read_csv('/Users/tylerbastian/Thesis/Psych Constructs
                      /hopelessnessControl.csv')
case_tweets = case['tweet']
control_tweets = control['tweet']
del(case, control)

def remove_emoji(string):
    emoji_pattern = re.compile("["
                               u"\U0001F600-\U0001F64F"  # emoticons
                               u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                               u"\U0001F680-\U0001F6FF"  # transport & map symbols
                               u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
```

```
                                u"\U00002500-\U00002BEF"  # chinese char
                                u"\U00002702-\U000027B0"
                                u"\U00002702-\U000027B0"
                                u"\U000024C2-\U0001F251"
                                u"\U0001f926-\U0001f937"
                                u"\U00010000-\U0010ffff"
                                u"\u2640-\u2642"
                                u"\u2600-\u2B55"
                                u"\u200d"
                                u"\u23cf"
                                u"\u23e9"
                                u"\u231a"
                                u"\ufe0f"  # dingbats
                                u"\u3030"
                                "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', string)

#Case tweet cleaning
text = []
for i in case_tweets:
    text.append(re.sub(r"(?:\@|https?\://)\S+", "", i))
text2 = []
for i in text:
    text2.append(re.sub(r"@\w+", "", i).replace("RT",''))
text3 = []
for i in text2:
    text3.append(re.sub(r"#\w+","",i))
text4 = []
for i in text3:
```

```
    text4.append(remove_emoji(i))
tokenizer = RegexpTokenizer(r'[a-zA-Z]+')
tweet_token = []
for x in text4:
    tweet_token.append(tokenizer.tokenize(x.lower()))

#Control tweet cleaning
text = []
for i in control_tweets:
    text.append(re.sub(r"(?:\@|https?\://)\S+", "", i))
text2 = []
for i in text:
    text2.append(re.sub(r"@\w+", "", i).replace("RT",''))
text3 = []
for i in text2:
    text3.append(re.sub(r"#\w+","",i))
text4 = []
for i in text3:
    text4.append(remove_emoji(i))
tokenizer = RegexpTokenizer(r'[a-zA-Z]+')
control_token = []
for x in text4:
    control_token.append(tokenizer.tokenize(x.lower()))

X = []
for words in control_token:
    X.append(' '.join(words))
Y = []
for words in tweet_token:
```

```
    Y.append(' '.join(words))

controls = pd.DataFrame(X)
controls.insert(0,'Sentiment','negative')
case = pd.DataFrame(Y)
case.insert(0,'Sentiment','positive')

#Graph proportionality of sentiments
tweets = pd.concat([case,controls])
import seaborn as sns
sns.countplot(x='Sentiment', data=tweets)
del(case, controls)
#Relatively equal proportion of control and case

X = tweets[0]
y = tweets['Sentiment']
y = np.array(list(map(lambda x: 1 if x=="positive" else 0, y)))
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=42)

tokenizer = Tokenizer(num_words = 5000)
tokenizer.fit_on_texts(X_train)
X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)

vocab_size = len(tokenizer.word_index)+1
maxlen = 100
X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
```

```
X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)

### Load Glove embeddings
from numpy import array
from numpy import asarray
from numpy import zeros

embeddings_dictionary = dict()
path = '/Users/tylerbastian/Desktop/Fall 2021/Thesis/Embeddings/glove.6B.100d.txt'
glove_file = open(path, encoding="utf8")

for line in glove_file:
    records = line.split()
    word = records[0]
    vector_dimensions = asarray(records[1:], dtype='float32')
    embeddings_dictionary [word] = vector_dimensions
glove_file.close()

embedding_matrix = zeros((vocab_size, 100))
for word, index in tokenizer.word_index.items():
    embedding_vector = embeddings_dictionary.get(word)
    if embedding_vector is not None:
        embedding_matrix[index] = embedding_vector


model=Sequential()
embedding_layer = Embedding(vocab_size, 100,
                            weights=[embedding_matrix],
                            input_length=maxlen,
```

```
                            trainable=False)
model.add(embedding_layer)
model.add(Conv1D(128, 5, activation='relu'))
model.add(GlobalMaxPool1D())
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy',
              metrics=['acc'])
print(model.summary())

history = model.fit(X_train, y_train, batch_size=128,
                    epochs = 6, verbose=1, validation_split=0.2)
score = model.evaluate(X_test, y_test, verbose=1)

print("Test Score:", score[0])
print("Test Accuracy:", score[1])

plt.subplot(211)
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.subplot(212)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
```

```
plt.legend(['train','test'], loc = 'upper left')
plt.tight_layout()

###Save the model
model.save('hopelessnessglovemodel.h5')
del(model, case_tweets, control_tweets,
    embedding_matrix, embeddings_dictionary,
    X_test, X_train, y_test, y_train)
```

B.6 Psychological Construct Classification

Once neural networks have been trained, evaluated, and all models have been saved we then classify each tweets psychological constructs with the following code:

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import re
import nltk
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from numpy import array
from keras.models import load_model
from keras.preprocessing.text import one_hot
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers.core import Activation, Dropout, Dense
from keras.layers import Flatten, Conv1D, LSTM
from keras.layers import GlobalMaxPool1D
from keras.layers.embeddings import Embedding
```

```
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer

def remove_emoji(string):
    emoji_pattern = re.compile("["
                               u"\U0001F600-\U0001F64F"  # emoticons
                               u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                               u"\U0001F680-\U0001F6FF"  # transport & map symbols
                               u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                               u"\U00002500-\U00002BEF"  # chinese char
                               u"\U00002702-\U000027B0"
                               u"\U00002702-\U000027B0"
                               u"\U000024C2-\U0001F251"
                               u"\U0001f926-\U0001f937"
                               u"\U00010000-\U0010ffff"
                               u"\u2640-\u2642"
                               u"\u2600-\u2B55"
                               u"\u200d"
                               u"\u23cf"
                               u"\u23e9"
                               u"\u231a"
                               u"\ufe0f"  # dingbats
                               u"\u3030"
                               "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', string)

#Load Psychological Construct Models
anxiety_model = load_model('/Users/tylerbastian/Desktop/Spring 2022/
                           Thesis/models/anxietyglovemodel.h5')
```

```
hopeless_model = load_model('/Users/tylerbastian/Desktop/Spring 2022/
                            Thesis/models/hopelessnessglovemodel.h5')
burden_model = load_model('/Users/tylerbastian/Desktop/Spring 2022/
                            Thesis/models/burdenglovemodel.h5')
lonely_model = load_model('/Users/tylerbastian/Desktop/Spring 2022/
                            Thesis/models/lonelyglovemodel.h5')
stress_model = load_model('/Users/tylerbastian/Desktop/Spring 2022/
                            Thesis/models/stressglovemodel.h5')
insomnia_model = load_model('/Users/tylerbastian/Desktop/Spring 2022/
                            Thesis/models/insomniaglovemodel.h5')
depression1_model = load_model('/Users/tylerbastian/Desktop/Spring 2022/
                                Thesis/models/depression1glovemodel.h5')
depression2_model = load_model('/Users/tylerbastian/Desktop/Spring 2022/
                                Thesis/models/depression2glovemodel.h5')
depression3_model = load_model('/Users/tylerbastian/Desktop/Spring 2022/
                                Thesis/models/depression3glovemodel.h5')


#Load the timeline data
predict_file = '/Users/tylerbastian/PycharmProjects/Stat766/Controltimeline2.csv'
data = pd.read_csv(predict_file)

text = []
for i in data['tweet']:
    text.append(re.sub(r"(?:\@|https?\://)\S+", "", i))
text2 = []
for i in text:
    text2.append(re.sub(r"@\w+", "", i).replace("RT",''))
text3 = []
```

```
for i in text2:
    text3.append(re.sub(r"#\w+","",i))
text4 = []
for i in text3:
    text4.append(remove_emoji(i))
tokenizer = RegexpTokenizer(r'[a-zA-Z]+')
tweet_token = []
for x in text4:
    tweet_token.append(tokenizer.tokenize(x.lower()))
Y = []
for words in tweet_token:
    Y.append(' '.join(words))

#del(data, predict_file)

#Collect anxiety predictions
import time
maxlen = 100
controlpred = pd.read_csv('/Users/tylerbastian/PycharmProjects/
                           Stat766/controltimelinepreds.csv')

####DONT RUN THESE AGAIN
controlpred = pd.DataFrame()
controlpred['author_id'] = data['author_id']
controlpred['created_at'] = data['created_at']

#Anxiety predictions
anxiety_preds = []
count = 0
```

```
for i in range(len(Y)):
    one = Y[i]
    count += 1
    tokenizer = Tokenizer(num_words=5000)
    tokenizer.fit_on_texts(one)
    instance = tokenizer.texts_to_sequences(one)
    flat_list = []
    for sublist in instance:
        for item in sublist:
            flat_list.append(item)
    flat_list = [flat_list]
    instance = pad_sequences(flat_list, padding='post', maxlen=maxlen)
    pred = anxiety_model.predict(instance)
    anxiety_preds.append(pred.item())
    print(count)
controlpred['Anxiety'] = anxiety_preds
del(anxiety_preds, anxiety_model)
controlpred.to_csv('controltimelinepreds.csv', index=False)



#predictions = pd.read_csv('/Users/tylerbastian/PycharmProjects/Stat766/
                           timelinepredictions.csv')
#Hopelessness predictions
hopeless_model = load_model('/Users/tylerbastian/Desktop/Fall 2021/
                           Thesis/models/hopelessnessglovemodel.h5')
hopeless_preds = []
count = 0
for i in range(len(Y)):
    one = Y[i]
```

```
    count += 1
    tokenizer = Tokenizer(num_words=5000)
    tokenizer.fit_on_texts(one)
    instance = tokenizer.texts_to_sequences(one)
    flat_list = []
    for sublist in instance:
        for item in sublist:
            flat_list.append(item)
    flat_list = [flat_list]
    instance = pad_sequences(flat_list, padding='post', maxlen=maxlen)
    pred = hopeless_model.predict(instance)
    hopeless_preds.append(pred.item())
    print(count)
controlpred['Hopelessness'] = hopeless_preds
del(hopeless_preds, hopeless_model)
controlpred.to_csv('controltimelinepreds.csv', index=False)

#Burden predictions
burden_model = load_model('/Users/tylerbastian/Desktop/Fall 2021/
                          Thesis/models/burdenglovemodel.h5')
burden_preds = []
count = 0
for i in range(len(Y)):
    one = Y[i]
    count += 1
    tokenizer = Tokenizer(num_words=5000)
    tokenizer.fit_on_texts(one)
    instance = tokenizer.texts_to_sequences(one)
    flat_list = []
```

```
    for sublist in instance:
        for item in sublist:
            flat_list.append(item)
    flat_list = [flat_list]
    instance = pad_sequences(flat_list, padding='post', maxlen=maxlen)
    pred = burden_model.predict(instance)
    burden_preds.append(pred.item())
    print(count)
controlpred['Burden'] = burden_preds
del(burden_preds, burden_model)
controlpred.to_csv('controltimelinepreds.csv', index=False)

#Lonely predictions
lonely_model = load_model('/Users/tylerbastian/Desktop/Fall 2021/
                          Thesis/models/lonelyglovemodel.h5')
lonely_preds = []
count = 0
for i in range(len(Y)):
    one = Y[i]
    count += 1
    tokenizer = Tokenizer(num_words=5000)
    tokenizer.fit_on_texts(one)
    instance = tokenizer.texts_to_sequences(one)
    flat_list = []
    for sublist in instance:
        for item in sublist:
            flat_list.append(item)
    flat_list = [flat_list]
    instance = pad_sequences(flat_list, padding='post', maxlen=maxlen)
```

```
    pred = lonely_model.predict(instance)
    lonely_preds.append(pred.item())
    print(count)
controlpred['Loneliness'] = lonely_preds
del(lonely_preds, lonely_model)
controlpred.to_csv('controltimelinepreds.csv', index=False)

#Stress predictions
stress_model = load_model('/Users/tylerbastian/Desktop/Fall 2021/
                          Thesis/models/stressglovemodel.h5')
stress_preds = []
count = 0
for i in range(len(Y)):
    one = Y[i]
    count += 1
    tokenizer = Tokenizer(num_words=5000)
    tokenizer.fit_on_texts(one)
    instance = tokenizer.texts_to_sequences(one)
    flat_list = []
    for sublist in instance:
        for item in sublist:
            flat_list.append(item)
    flat_list = [flat_list]
    instance = pad_sequences(flat_list, padding='post', maxlen=maxlen)
    pred = stress_model.predict(instance)
    stress_preds.append(pred.item())
    print(count)
controlpred['Stress'] = stress_preds
del(stress_preds, stress_model)
```

```
controlpred.to_csv('controltimelinepreds.csv', index=False)

#Insomnia predictions
insomnia_model = load_model('/Users/tylerbastian/Desktop/Fall 2021/
                            Thesis/models/insomniaglovemodel.h5')
insomnia_preds = []
count = 0
for i in range(len(Y)):
    one = Y[i]
    count += 1
    tokenizer = Tokenizer(num_words=5000)
    tokenizer.fit_on_texts(one)
    instance = tokenizer.texts_to_sequences(one)
    flat_list = []
    for sublist in instance:
        for item in sublist:
            flat_list.append(item)
    flat_list = [flat_list]
    instance = pad_sequences(flat_list, padding='post', maxlen=maxlen)
    pred = insomnia_model.predict(instance)
    insomnia_preds.append(pred.item())
    print(count)
controlpred['Insomnia'] = insomnia_preds
del(insomnia_preds, insomnia_model)
controlpred.to_csv('controltimelinepreds.csv', index=False)

#Depression 1 predictions
depression1_model = load_model('/Users/tylerbastian/Desktop/Fall 2021/
                               Thesis/models/depression1glovemodel.h5')
```

```
dep1_preds = []
count = 0
for i in range(len(Y)):
    one = Y[i]
    count += 1
    tokenizer = Tokenizer(num_words=5000)
    tokenizer.fit_on_texts(one)
    instance = tokenizer.texts_to_sequences(one)
    flat_list = []
    for sublist in instance:
        for item in sublist:
            flat_list.append(item)
    flat_list = [flat_list]
    instance = pad_sequences(flat_list, padding='post', maxlen=maxlen)
    pred = depression1_model.predict(instance)
    dep1_preds.append(pred.item())
    print(count)
controlpred['Depression 1'] = dep1_preds
del(dep1_preds, depression1_model)
controlpred.to_csv('controltimelinepreds.csv', index=False)

#Depression 2 predictions
depression2_model = load_model('/Users/tylerbastian/Desktop/Fall 2021/
                                Thesis/models/depression2glovemodel.h5')
dep2_preds = []
count = 0
for i in range(len(Y)):
    one = Y[i]
    count += 1
```

```
    tokenizer = Tokenizer(num_words=5000)
    tokenizer.fit_on_texts(one)
    instance = tokenizer.texts_to_sequences(one)
    flat_list = []
    for sublist in instance:
        for item in sublist:
            flat_list.append(item)
    flat_list = [flat_list]
    instance = pad_sequences(flat_list, padding='post', maxlen=maxlen)
    pred = depression2_model.predict(instance)
    dep2_preds.append(pred.item())
    print(count)
controlpred['Depression 2'] = dep2_preds
del(dep2_preds, depression2_model)
controlpred.to_csv('controltimelinepreds.csv', index=False)

#Depression 3 predictions
depression3_model = load_model('/Users/tylerbastian/Desktop/Fall 2021/
                                Thesis/models/depression3glovemodel.h5')
dep3_preds = []
count = 0
for i in range(len(Y)):
    one = Y[i]
    count += 1
    tokenizer = Tokenizer(num_words=5000)
    tokenizer.fit_on_texts(one)
    instance = tokenizer.texts_to_sequences(one)
    flat_list = []
    for sublist in instance:
```

```
        for item in sublist:
            flat_list.append(item)
    flat_list = [flat_list]
    instance = pad_sequences(flat_list, padding='post', maxlen=maxlen)
    pred = depression3_model.predict(instance)
    dep3_preds.append(pred.item())
    print(count)
controlpred['Depression 3'] = dep3_preds
del(dep3_preds, depression3_model)
controlpred['text'] = data['tweet']
controlpred.to_csv('controltimelinepreds.csv', index=False)

import textblob
from textblob import TextBlob
def getSubjectivity(review):
    return TextBlob(review).sentiment.subjectivity
# function to calculate polarity
def getPolarity(review):
        return TextBlob(review).sentiment.polarity
tweets = pd.DataFrame(Y)
controlpred['Subjectivity'] = tweets[0].apply(getSubjectivity)
controlpred['Polarity'] = tweets[0].apply(getPolarity)
controlpred.to_csv('controltimelinepreds.csv', index=False)
```

B.7 Classify SI Tweets

After each tweet has been classified with all psychological constructs we then read each tweet and assign it a 0 for accounts not showing signs of SI and 1 for accounts showing signs of SI using the following code.

```
import pandas as pd
import re
from pandas import read_csv
from nltk.tokenize import RegexpTokenizer
import plotly.graph_objects as go
import datetime
import matplotlib
import matplotlib.pyplot as plt
from textblob import TextBlob
import numpy as np

#Functions for data preperation
# function to calculate subjectivity
def getSubjectivity(review):
    return TextBlob(review).sentiment.subjectivity
# function to calculate polarity
def getPolarity(review):
        return TextBlob(review).sentiment.polarity
def remove_emoji(string):
    emoji_pattern = re.compile("["
                               u"\U0001F600-\U0001F64F"  # emoticons
                               u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                               u"\U0001F680-\U0001F6FF"  # transport & map symbols
                               u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                               u"\U00002500-\U00002BEF"  # chinese char
                               u"\U00002702-\U000027B0"
                               u"\U00002702-\U000027B0"
                               u"\U000024C2-\U0001F251"
                               u"\U0001f926-\U0001f937"
```

```
                                    u"\U00010000-\U0010ffff"
                                    u"\u2640-\u2642"
                                    u"\u2600-\u2B55"
                                    u"\u200d"
                                    u"\u23cf"
                                    u"\u23e9"
                                    u"\u231a"
                                    u"\ufe0f"  # dingbats
                                    u"\u3030"
                                    "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', string)

#Data preperation
timeline = read_csv('/Users/tylerbastian/Thesis/Psych Constructs/timeline.csv')
case = read_csv('/Users/tylerbastian/PycharmProjects/Stat766/timelinescores.csv')
cons = read_csv('/Users/tylerbastian/PycharmProjects/Stat766/controlscores.csv')

case = pd.DataFrame(case)
cons = pd.DataFrame(cons)
case['text'] = timeline['text']
data = pd.concat([case,cons])
data.info()
del(timeline,case,cons)

text = []
for i in data['text']:
    text.append(re.sub(r"(?:\@|https?\://)\S+", "", i))
text2 = []
for i in text:
```

```
    text2.append(re.sub(r"@\w+", "", i).replace("RT",''))
text3 = []
for i in text2:
    text3.append(re.sub(r"#\w+","",i))
text4 = []
for i in text3:
    text4.append(remove_emoji(i))
tokenizer = RegexpTokenizer(r'[a-zA-Z]+')
tweet_token = []
for x in text4:
    tweet_token.append(tokenizer.tokenize(x.lower()))
Y = []
for words in tweet_token:
    Y.append(' '.join(words))

data['Text'] = Y
del(text, text2, text3, text4,tweet_token, words, x, i, Y)
#Break down date and time into individual variables
#data['Year'] = pd.to_datetime(data['created_at']).dt.year
#data['Month'] = pd.to_datetime(data['created_at']).dt.month
#data['Day'] = pd.to_datetime(data['created_at']).dt.day
#data['Hour'] = pd.to_datetime(data['created_at']).dt.hour
#data['Minute'] = pd.to_datetime(data['created_at']).dt.minute
#data['Second'] = pd.to_datetime(data['created_at']).dt.second
#Calculate subjectivity and polarity scores using TextBlob
data['Subjectivity'] = data['Text'].apply(getSubjectivity)
data['Polarity'] = data['Text'].apply(getPolarity)
cols = ['author_id','created_at','Anxiety','Hopelessness',
            'Burden','Loneliness','Stress',
```

```
            'Insomnia','Depression 1','Depression 2',
            'Depression 3','Subjectivity','Polarity','Text']
data2 = data[cols]

data2.to_csv('RandomForestSI.csv', index=False)
#Identify suciidal ideation using phrases from SI tweet criterion
# #and Past SA or suicide plan tweet criteria
del(data,cols)

df = read_csv('/Users/tylerbastian/PycharmProjects/Stat766/RandomForestSI.csv')
text = list(df['Text'])
data = pd.DataFrame(text)
data['author_id'] = df['author_id']
data = data.fillna('None')
text = list(data[0])
tweets = tweet_token[:1000]

SIphrases = [['i','am','planning','to','die'],
             ['i','am','going','to','commit','suicide'],
             ['i','attempted','suicide'],
             ['i','planning','to','end','my','life'],
             ['i','planned','to','end','my','life'],
             ['i','planned','die'],
             ['i','planning','to','kill','myself'],
             ['i','tried','to','kill','myself'],
             ['i','thinking','suicide'],
             ['i',"can't",'go','on','living'],
             ['i','want','to','die'],
             ['i','want','to','kill','myself'],
```

```
            ['i','going','kill','myself'],
            ['i','have','plan','kill','myself'],
            ['i','contemplating','suicide'],
            ['i','planning','suicide'],
            ['i','feeling','suicidal'],
            ['i','having','suicidal','thoughts'],
            ['i','want','to','commit','suicide'],
            ['i','want','to','end','my','life'],
            ['i','thinking','how','to','kill','myself']]

pastSI = [['i','am','planning','to','die'],
          ['i','attempted','suicide'],
          ['my','suicide','attempt'],
          ['i','planning','to','end','my','life'],
          ['i','planned','to','end','my','life'],
          ['i','planned','die'],
          ['i','planning','to','kill','myself'],
          ['i','tried','to','kill','myself'],
          ['i','have','plan','kill','myself'],
          ['i','planning','suicide']]

def phrase_finder(lst, phrases):
    result = []
    for i in range(len(lst)):
        tweet = str(lst[i])
        if all(word in tweet for word in phrases):
            result.append(i)
        else:
            pass
```

```
    return(result)

SItweets = []
for i in range(len(SIphrases)):
    results=phrase_finder(Y,SIphrases[i])
    SItweets.append(results)

indices = [item for sublist in SItweets for item in sublist]

unique = []
for i in range(len(indices)):
    if indices[i] not in unique:
        unique.append(indices[i])
    else:
        pass
unique.sort()

Y = [0]*len(df)
for i in unique:
    Y[i] = 1
df['Y'] = Y

if any(unique) == 2:
    print('TRUE')
else:
    print('FALSE')

df2 = pd.DataFrame(text)
df2['Y'] = Y
```

```
TPSI = read_csv('/Users/tylerbastian/Desktop/Fall 2021/Thesis/Rows.csv')
for i in range(len(TPSI)):
    row = df2[:i]

#We need the last months worth of data from each author id.

df = read_csv('/Users/tylerbastian/PycharmProjects/Stat766/RandomForestSI.csv')
data = df[:100]
```

B.8 Random Forest SI Classification Training and Evaluation

After assigning tweets as showing signs of SI we then train 4 Random Forest models with aggregated data to predict suicidal ideation 4, 7, 14, and 21 days in advance.

```
import pandas as pd
from pandas import read_csv
import random
import datetime
from datetime import timedelta
from dateutil.relativedelta import relativedelta

SIs = read_csv('/Users/tylerbastian/PycharmProjects/Stat766/RandomForestSI.csv')
len(SIs.author_id.unique())
control1 = read_csv('/Users/tylerbastian/PycharmProjects/Stat766/controltimelinepreds.cs

Y = [0]*len(SIs)
```

```
TPSI = read_csv('/Users/tylerbastian/Desktop/Spring 2022/Thesis/Rows.csv', header=None)
for i in TPSI[0]:
    Y[i] = 1
SIs['Y'] = Y
SIcons = SIs.loc[SIs.groupby('author_id')['Y'].transform(sum)==0]
SIs = SIs.loc[SIs.groupby('author_id')['Y'].transform(sum)>=1]

Y = [0] * len(control1)
control1['Y'] = Y

unique = []
for i in control1['author_id']:
    if i not in unique:
        unique.append(i)
for i in SIcons['author_id']:
    if i not in unique:
        unique.append(i)

len(unique)

#to have 9% of our data be cases we need 3800 users in total
n = 3800 - 342
randusers = random.sample(unique,n)

control1.columns
control2 = control1[['author_id','created_at','Anxiety','Hopelessness','Burden',
                     'Loneliness','Stress','Insomnia','Depression 1','Depression 2',
                     'Depression 3','Subjectivity','Polarity','Y']]
SIcons2 = SIcons[['author_id','created_at','Anxiety','Hopelessness','Burden',
```

```
                    'Loneliness','Stress','Insomnia','Depression 1','Depression 2',
                    'Depression 3','Subjectivity','Polarity','Y']]
SIs = SIs[['author_id','created_at','Anxiety','Hopelessness','Burden',
                    'Loneliness','Stress','Insomnia','Depression 1','Depression 2',
                    'Depression 3','Subjectivity','Polarity','Y']]
del(SIcons)
Controls2 = pd.concat([control2, SIcons2])
###### We have all our controls in one dataframe and our cases in another dataframe.

Controls = Controls2[Controls2.author_id.isin(randusers)]

unique = []
for i in Controls['author_id']:
    if i not in unique:
        unique.append(i)
len(unique)

##Aggregate function
dates = []
for i in range(len(SIs)):
    row = SIs.iloc[i]
    if row['Y'] == 1:
        dates.append(row['created_at'])



dates = list(pd.to_datetime(dates, format = '%Y-%m-%d %H:%M:%S'))
SIdates = [i.strftime('%Y-%m-%d %H:%M:%S') for i in dates]
SIdates = pd.DataFrame(pd.to_datetime(SIdates, format = '%Y-%m-%d %H:%M:%S'))
SIdates['end'] = SIdates[0] - datetime.timedelta(days=7)
```

```
SIstart = list(SIdates[0])
SIend = list(SIdates['end'])
dates = pd.to_datetime(SIs['created_at'], format = '%Y-%m-%d %H:%M:%S')
dates = [i.strftime('%Y-%m-%d %H:%M:%S') for i in dates]
SIs['created_at'] = dates
SIs['created_at'] = pd.to_datetime(SIs['created_at'], format= '%Y-%m-%d %H:%M:%S')



SIusers = []
for i in SIs['author_id']:
    if i not in SIusers:
        SIusers.append(i)
ndays=7
def aggregate_users(ndays, SIs):
    aggregated = pd.DataFrame(columns=['Anxiety','Hopelessness','Burden','Loneliness','S
                                       'Insomnia','Depression 1','Depression 2','Depression
                                       'Subjectivity','Polarity'])
    for i in SIusers:
        df = pd.DataFrame(SIs[SIs['author_id'] == i])
        for j in range(len(df)):
            row = df.iloc[j]
            if row['Y'] == 1:
                date = row['created_at']
                end = date-datetime.timedelta(days=ndays)
                mask = (df['created_at']<=date)&(df['created_at']>=end)
                a = df.loc[mask]
                a = a.drop_duplicates()
                a = a[['Anxiety','Hopelessness','Burden','Loneliness','Stress',
                                       'Insomnia','Depression 1','Depression 2','Depression
```

```
                                    'Subjectivity','Polarity']]
                agg = pd.DataFrame(a.mean(axis=0)).transpose()
                aggregated = pd.concat([aggregated,agg])
    return(aggregated)
aggSI4 = aggregate_users(4, SIs)
aggSI7 = aggregate_users(7, SIs)
aggSI14 = aggregate_users(14, SIs)
aggSI21 = aggregate_users(21, SIs)
aggSI4.to_csv('aggregateSI4days.csv', index=False)
aggSI7.to_csv('aggregateSI7days.csv', index=False)
aggSI14.to_csv('aggregateSI14days.csv', index=False)
aggSI21.to_csv('aggregateSI21days.csv', index=False)


Controlusers = []
for i in Controls['author_id']:
    if i not in Controlusers:
        Controlusers.append(i)
len(Controlusers)

Controls = Controls2[Controls2.author_id.isin(randusers)]
Controls = Controls.reset_index(drop=True)
dates = list(Controls['created_at'])
dates = [i.replace('+00:00', '') for i in dates]
controldates = pd.DataFrame(pd.to_datetime(dates, format = '%Y-%m-%d %H:%M:%S'))
Controls['created_at'] = controldates
Controls['created_at'] = pd.to_datetime(Controls['created_at'], format = '%Y-%m-%d %H:%M
```

```
def aggregate_controls(ndays, Controls):
    aggregatedControls = pd.DataFrame(columns=['Anxiety', 'Hopelessness', 'Burden', 'Lon
                                        'Insomnia', 'Depression 1', 'Depression 2', 'Depr
                                        'Subjectivity', 'Polarity'])
    for i in Controlusers:
        df = pd.DataFrame(Controls[Controls['author_id'] == i])
        row = df.sample()
        date = row['created_at']
        end = date - datetime.timedelta(days=ndays)
        date = [i.strftime('%Y-%m-%d %H:%M:%S') for i in date]
        end = [i.strftime('%Y-%m-%d %H:%M:%S') for i in end]
        mask = (df['created_at']==date[0])&(df['created_at']>=end[0])
        a = df.loc[mask]
        a = a.drop_duplicates()
        a = a[['Anxiety', 'Hopelessness', 'Burden', 'Loneliness', 'Stress',
                   'Insomnia', 'Depression 1', 'Depression 2', 'Depression 3',
                   'Subjectivity', 'Polarity']]
        agg = pd.DataFrame(a.mean(axis=0)).transpose()
        aggregatedControls = pd.concat([aggregatedControls, agg])
    return(aggregatedControls)

aggC4 = aggregate_controls(4, Controls)
aggC7 = aggregate_controls(7, Controls)
aggC14 = aggregate_controls(14, Controls)
aggC21 = aggregate_controls(21, Controls)
aggC4.to_csv('aggC4.csv', index=False)
aggC7.to_csv('aggC7.csv', index=False)
aggC14.to_csv('aggC14.csv', index=False)
```

```
aggC21.to_csv('aggC21.csv', index=False)



from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
import matplotlib.pyplot as plt
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import PrecisionRecallDisplay
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import auc
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, Con
from collections import Counter

aggC4Y = [0] * len(aggC4)
aggSI4Y = [1] * len(aggSI4)
Y4 = np.concatenate([aggC4Y, aggSI4Y])
data4 = np.concatenate([aggC4, aggSI4])
x_train4, x_test4, y_train4, y_test4 = train_test_split(data4, Y4, test_size=0.33)

aggC7Y = [0] * len(aggC7)
aggSI7Y = [1] * len(aggSI7)
Y7 = np.concatenate([aggC7Y, aggSI7Y])
data7 = np.concatenate([aggC7, aggSI7])
x_train7, x_test7, y_train7, y_test7 = train_test_split(data7, Y7, test_size=0.33)

aggC14Y = [0] * len(aggC14)
aggSI14Y = [1] * len(aggSI14)
```

```
Y14 = np.concatenate([aggC14Y, aggSI14Y])
data14 = np.concatenate([aggC14, aggSI14])
x_train14, x_test14, y_train14, y_test14 = train_test_split(data14, Y14, test_size=0.33)

aggC21Y = [0] * len(aggC21)
aggSI21Y = [1] * len(aggSI21)
Y21 = np.concatenate([aggC21Y, aggSI21Y])
data21 = np.concatenate([aggC21, aggSI21])
x_train21, x_test21, y_train21, y_test21 = train_test_split(data21, Y21, test_size=0.33)

###RANDOM FOREST

clf4 = RandomForestClassifier(n_estimators=10, random_state=42)
clf4.fit(x_train4, y_train4)
y_pred4 = clf4.predict(x_test4)
print('Accuracy:', metrics.accuracy_score(y_test4, y_pred4))
feature_imp4 = pd.Series(clf4.feature_importances_, index= ['Anxiety', 'Hopelessness', '
                    'Insomnia', 'Depression 1', 'Depression 2', 'Depression 3',
                    'Subjectivity', 'Polarity']).sort_values(ascending=False)
print(feature_imp4)

clf7 = RandomForestClassifier(n_estimators=10, random_state=42)
clf7.fit(x_train7, y_train7)
y_pred7 = clf7.predict(x_test7)
print('Accuracy:', metrics.accuracy_score(y_test7, y_pred7))
feature_imp7 = pd.Series(clf7.feature_importances_, index= ['Anxiety', 'Hopelessness', '
                    'Insomnia', 'Depression 1', 'Depression 2', 'Depression 3',
                    'Subjectivity', 'Polarity']).sort_values(ascending=False)
print(feature_imp7)
```

```
clf14 = RandomForestClassifier(n_estimators=10, random_state=42)
clf14.fit(x_train14, y_train14)
y_pred14 = clf14.predict(x_test14)
print('Accuracy:', metrics.accuracy_score(y_test14, y_pred14))
feature_imp14 = pd.Series(clf14.feature_importances_, index= ['Anxiety', 'Hopelessness',
                    'Insomnia', 'Depression 1', 'Depression 2', 'Depression 3',
                    'Subjectivity', 'Polarity']).sort_values(ascending=False)
print(feature_imp14)

clf21 = RandomForestClassifier(n_estimators=10, random_state=42)
clf21.fit(x_train21, y_train21)
y_pred21 = clf21.predict(x_test21)
print('Accuracy:', metrics.accuracy_score(y_test21, y_pred21))
feature_imp21 = pd.Series(clf21.feature_importances_, index=['Anxiety', 'Hopelessness',
                    'Insomnia', 'Depression 1', 'Depression 2', 'Depression 3',
                    'Subjectivity', 'Polarity']).sort_values(ascending=False)
print(feature_imp21)



ax = plt.gca()
y_pred_proba4 = clf4.predict_proba(x_test4)[::,1]
y_pred_proba7 = clf7.predict_proba(x_test7)[::,1]
y_pred_proba14 = clf14.predict_proba(x_test14)[::,1]
y_pred_proba21 = clf21.predict_proba(x_test21)[::,1]
fpr4, tpr4, _ = metrics.roc_curve(y_test4, y_pred_proba4)
fpr7, tpr7, _ = metrics.roc_curve(y_test7, y_pred_proba7)
fpr14, tpr14, _ = metrics.roc_curve(y_test14, y_pred_proba14)
fpr21, tpr21, _ = metrics.roc_curve(y_test21, y_pred_proba21)
```

```
plt.plot(fpr4,tpr4, label = "4 Days")
plt.plot(fpr7, tpr7, label = '7 Days')
plt.plot(fpr14,tpr14, label = "14 Days")
plt.plot(fpr21, tpr21, label = "21 Days")
plt.legend(loc='lower right')
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.title("ROC Curve")
plt.savefig("SIRF.png")



#Precision-recall

clfprob4 = clf4.predict_proba(x_test4)[:,1]
precision4, recall4, _ = precision_recall_curve(y_test4, clfprob4)
auc_rf4 = auc(recall4, precision4)

clfprob7 = clf7.predict_proba(x_test7)[:,1]
precision7, recall7, _ = precision_recall_curve(y_test7, clfprob7)
auc_rf7 = auc(recall7, precision7)

clfprob14 = clf14.predict_proba(x_test14)[:,1]
precision14, recall14, _ = precision_recall_curve(y_test14, clfprob14)
auc_rf14 = auc(recall14, precision14)

clfprob21 = clf21.predict_proba(x_test21)[:,1]
precision21, recall21, _ = precision_recall_curve(y_test21, clfprob21)
```

```
auc_rf21 = auc(recall21, precision21)



plt.figure(figsize=(12,7))
plt.plot(recall4, precision4, label = f'AUC 4 Days = {auc_rf4:.2f}')
plt.plot(recall7, precision7, label = f'AUC 7 Days = {auc_rf7:.2f}')
plt.plot(recall14, precision14, label = f'AUC 14 Days = {auc_rf14:.2f}')
plt.plot(recall21, precision21, label = f'AUC 21 Days = {auc_rf21:.2f}')
plt.legend()
plt.ylabel("Precision")
plt.xlabel("Recall")
plt.title("Precision-Recall")
plt.show()
plt.savefig("PrecisionRecall.png")



##Confusion Matrices
cm4 = confusion_matrix(y_test4, y_pred4, labels=clf4.classes_)
CM4 = ConfusionMatrixDisplay(confusion_matrix=cm4, display_labels=clf4.classes_)
CM4.plot()
plt.savefig('CM4.png')

cm7 = confusion_matrix(y_test7, y_pred7, labels=clf7.classes_)
CM7 = ConfusionMatrixDisplay(confusion_matrix=cm7, display_labels=clf7.classes_)
CM7.plot()
plt.savefig('CM7.png')

cm14 = confusion_matrix(y_test14, y_pred14, labels=clf14.classes_)
```

```
CM14 = ConfusionMatrixDisplay(confusion_matrix=cm14, display_labels=clf14.classes_)
CM14.plot()
plt.savefig('CM14.png')

cm21 = confusion_matrix(y_test21, y_pred21, labels=clf21.classes_)
CM21 = ConfusionMatrixDisplay(confusion_matrix=cm21, display_labels=clf21.classes_)
CM21.plot()
plt.savefig('CM21.png')

##plot single decision tree
from sklearn import tree
plt.figure(figsize=(10,20))
tree.plot_tree(clf21.estimators_[0], feature_names=feature_imp21.index, filled=True)
plt.savefig('SItree.png')



from textblob import TextBlob
print(TextBlob.__version__())

features = pd.DataFrame(pd.concat([feature_imp4, feature_imp7, feature_imp14, feature_im
features.columns = ["4 Days", "7 Days", "14 Days", "21 Days"]
features['Cons'] = features.index
plt.figure(figsize=(13,7))
plt.plot(features['Cons'], features['4 Days'], label = f'4 Days')
plt.plot(features['Cons'], features['7 Days'], label = f'7 Days')
plt.plot(features['Cons'], features['14 Days'], label = f'14 Days')
plt.plot(features['Cons'], features['21 Days'], label = f'21 Days')
plt.legend()
```

```
plt.title("Feature Importance")
plt.xlabel("Features")
plt.ylabel("Importance")
plt.show()
plt.savefig("FeatureImportance.png")
```