Unveiling Online Stress: Predictive Analysis of Social Media Content

Academic Year 2023/24 Term 1

IS460 – Machine Learning & Applications

Prepared for Dr Wang Zhaoxia

Group 1

Submitted by:

Full Name	Campus ID
Jaron Chan Zhuo Han	01393785
Adrianus Tjoatja Widjaja	01391444
Ramasamy Vighnesh	01394515
Ivan Yeo Wen Quan	01399621
Davin Shee	01392785

Table of Contents

1. Introduction

1.1 Problem Statement

Survey data shows stress levels in Singapore remain significantly higher compared to the global average, with 86% of Singapore respondents saying they are stressed and 15% saying they struggle to cope with stress. (Kalra et al., 2023) Furthermore, there has been a rise of social media rage or otherwise called venting, especially during the COVID-19 pandemic. (Factory, 2023) These ventings were done for attention or validation, emotional release, or due to the pressure for people to share their feelings. (Foundation, 2023) To address this problem, binary classification models can be used for early identification of stress levels in people based on their social media posts.

1.2 Motivation

The motivation behind our early detection of stress is based on two factors: To provide an optimal solution for stress detection, and to improve the public's mental health and wellbeing.

In the government and healthcare's perspective, doing a health screening for every individual to identify stress levels for the population is inefficient and expensive. Doing random sampling to estimate this figure, though may be cheaper and more realistic, also has its own limitations due to the nature of random sampling or the representativeness of the sample.

With the extensive utilisation of social media in Singapore and the fact that whether consciously or unconsciously, social media posts show signs of whether the author is stressed, we can take these posts to train up models to help us predict/classify whether one is stressed or not.

In the context of healthcare, this would reduce inaccuracies of traditional estimations and make it cheaper and more efficient than conducting health screenings for every individual for stress detection. An early detection would also help to prevent stress from escalating into more serious mental health issues, improve the mental well-being of the people living in Singapore and help create new opportunities to offer assistance and resources to help overcome stress.

2. Literature Review

2.1 Machine Learning Driven Mental Stress Detection on Reddit Posts using Natural Language Processing (Inamdar et al., 2023)

This journal paper is about stress detection using Natural Language Processing (NLP). The dataset that they used is the Dreddit dataset. The dataset utilised 2800 unique texts for training. Since the input dataset has less than 10000 observations, therefore traditional Machine Learning models were used rather than Deep Learning techniques which work better on large datasets. The journal paper utilised many embedding methods which included the use of Bidirectional Encoder Representations from Transformers, known as BERT, Embeddings from Language Models, known as ELMo, and Bag of Words. The Machine Learning models they used included Logistic Regression, XGBoost and Support Vector Machines (SVM).

Precision, Recall and F1 scores for our various experimental models

Embedding Method	Classifier	F1 score	Recall	Average precision	Accuracy
BERT	Logistic Regression	0.56	0.53	0.55	0.53
	XGBoost	0.68	0.53	0.56	0.56
	SVM	0.64	0.56	0.57	0.57
BoW*TF-IDF	Logistic Regression	0.70	0.60	0.64	0.65
	XGBoost	0.60	0.66	0.63	0.66
	SVM	0.69	0.64	0.60	0.64
ELMO	Logistic Regression	0.76	0.74	0.70	0.74
	XGBoost	0.70	0.61	0.62	0.63
	SVM	0.76	0.74	0.70	0.74

Fig 01: Performance Metric for the different embedding methods with ML models

Figure 01 above shows the results that the journal paper obtained. From Figure 01, we can see that ELMo embeddings was the best embedding method, with Logistic Regression and SVM performing equally, both performing better than XGBoost. However, there is a fundamental limitation which relates to the small dataset size. The small dataset size proved to be a fundamental limitation in achieving high accuracy for the predictive models. Furthermore, since the data is only obtained from a single source of social media platform which proved to be accurate for the given post, analysing posts made by authors on multiple platforms can give us a better sense of the mental state of the authors of the post. With that said, the insights and lessons learnt from this journal paper is that we can utilise a bigger dataset by obtaining and scraping data from multiple social media platforms to overcome the limitation which this journal paper faced.

2.2 StressSense: Stress Detection System (Thisishusseinali, 2023)

The next Literature Review is on a Kaggle solution titled “StressSense, Stress Detection System” and was created by the user “thisishusseinali” in 2023. This solution utilised the dataset from Kaggle titled “Human Stress Prediction”. We noted that this solution did not utilise any embedding methods, but did use many models within the solution. The Machine Learning models include the Decision Tree Classifier, Random Forest Classifier, AdaBoost Classifier, K-Neighbors Classifier, Stochastic Gradient Descent (SGD) Classifier, Multi-Layer Perceptron (MLP) Classifier, Multinomial Naïve Bayes, Bernoulli Naïve Bayes and Support Vector Classification which is an implementation of Support Vector Machines.

######-Model => BernoulliNB
Test Accuracy: 77.29%
Classification Report

	precision	recall	f1-score	support
No Stress	0.77	0.71	0.74	255
Stress	0.78	0.83	0.80	313
accuracy			0.77	568
macro avg	0.77	0.77	0.77	568
weighted avg	0.77	0.77	0.77	568

######-Model => MultinomialNB
Test Accuracy: 78.70%
Classification Report

	precision	recall	f1-score	support
No Stress	0.80	0.70	0.75	255
Stress	0.78	0.86	0.82	313
accuracy			0.79	568
macro avg	0.79	0.78	0.78	568
weighted avg	0.79	0.79	0.79	568

######-Model => MLPClassifier
Test Accuracy: 73.77%
Classification Report

	precision	recall	f1-score	support
No Stress	0.70	0.74	0.72	255
Stress	0.78	0.74	0.76	313
accuracy			0.74	568
macro avg	0.74	0.74	0.74	568
weighted avg	0.74	0.74	0.74	568

Fig 02, 03, 04: Performance Metrics for the top 3 performing models

As shown from Figures 02 to 04, the top 3 models identified are the Multinomial Naïve Bayes, Bernoulli Naïve Bayes and the Multi-Layer Perceptron (MLP) Classifier. However, as noted earlier, a limitation to this solution created by thisishusseinali on Kaggle is the lack of embedding methods. Such a limitation may have limited the overall performance and effectiveness of all of the machine learning models used. As such, the insights derived from this literature review would be to explore and use possible embedding techniques as previously seen in Literature Review #1. Another insight gained would be the possible use of ensemble learning to be able to combine the models since there are many machine learning models to use.

2.3 Stress Detection Methodology based on Social Media Network: A Proposed Design (Chaware et al., 2020)

For the last literature review, Literature Review #3, we will be examining a journal paper which also aims to detect stress to avoid further consequences such as going into depression, self-harming acts, etc. In this journal paper, they covered data solely from Facebook posts.

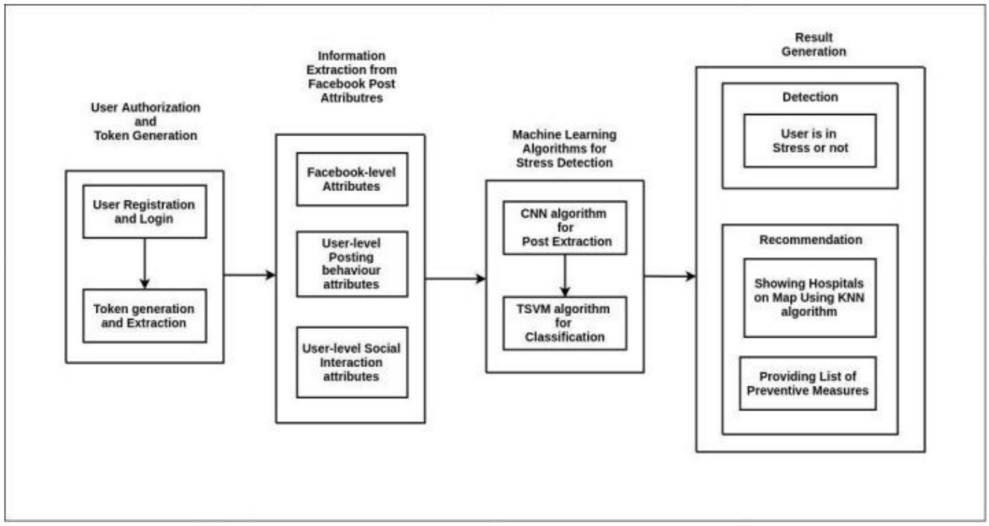


Fig 05: Proposed Design for stress detection workflow in Facebook posts

As seen in Figure 05, the authors of the journal paper decided to extract relevant information from the Facebook posts attributes which will then be used as input for their machine learning models. The machine learning models which this journal paper utilised include the use of Transductive Support Vector Machine (TSVM), Support Vector Machine (SVM), Naïve Bayes, Random Forest, Decision Tree, Ada-boosted Decision Tree. Once again, we also noted that this journal paper did not cover any embedding methods.

Sequence No	Algorithm used with Accuracy	
	Algorithm	*Accuracy*
1	TSVM	84.2%
2	SVM	82%
4	Naïve Bayes	77.5
5	Random Forest	73.8
6	Decision Tree	72.5
3	Adaboosted D-Tree	67%

Fig 06: Results of the various machine learning models utilised

The above Figure 06 showcases the results of the different machine learning models utilised to analyse the Facebook posts attributes. As seen, the only performance metric used was Accuracy, and the best

performing model was TSVM with an accuracy of 84.2%. The other algorithms like Decision Tree, Naïve Bayes, Random Forest failed to achieve expected accuracy since they gave an approximate accuracy of 70%.

From this journal paper, they noted that they had challenges in accurately interpreting and analysing the emotional nuances and complexities within the attributes of the Facebook posts which could have limited their models' performances and lead to potential inaccuracies in stress detection. The insights gained from this literature review would be to use embedding methods to help capture the nuances within the attributes of social media posts, and to also use cross-validation techniques in training and testing of our machine learning models.

3. Dataset

3.1 Raw Dataset

Initially, our team sourced our dataset from a Kaggle problem named “Human Stress Prediction ", which had 2838 observations and 120124 words. After the midterm presentation, our team acknowledged that we wanted to increase the robustness of the dataset as well as have social media comments from other social media applications. We found the solution to this in the form of a Github repository named “stress-detection”, which also contained tweets from Twitter(X). After merging and cleaning both datasets, the resultant dataset had 14364 observations with 518895 words. Regarding the split of the data, Label 0 (not stressed) had 7101 observations while Label 1 (stressed) had 7263 observations.

As shown in Table 1 below, for each row/observation, there are 2 features and 1 target variable.

Feature	Description
Categorization	Unique identifier of the comment
Body	The actual text sourced from both datasets mentioned above
Label	Boolean value representing whether the user typing the Body is stressed or not

Table 1: Description of features in the dataset

It was also interesting to see how the addition of the new dataset (including data from Twitter) changed some dominant words in the word cloud. For example, “excited” and "happy” became the dominant words for not stress, compared to “know” and “time” previously.

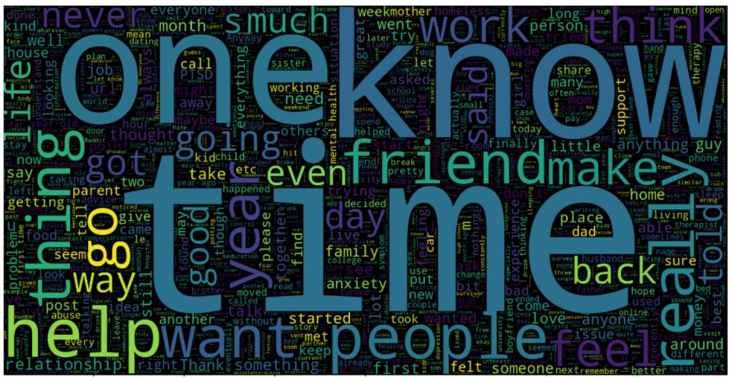


Fig 07: Word Cloud for “Not Stressed” (Before combining dataset)

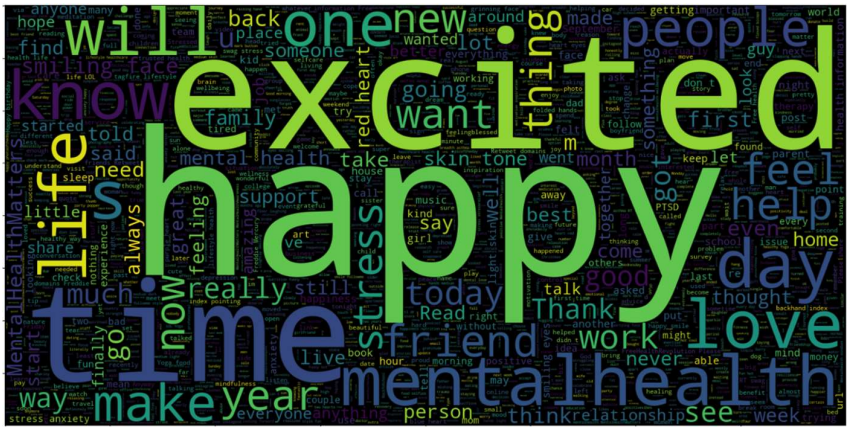


Fig 08: Word Cloud for "Not Stressed" (After combining dataset)

3.2 Exploratory Data Analysis

3.2.1 Text Length Analysis (Number of characters in each text)

We computed the number of characters present in each text entry within our dataset to understand the length of text in terms of characters. This can provide valuable insights into the variation across different samples. Examining this metric helps us identify outliers, such as unusually short or exceptionally long texts, which might have significance in certain analyses. By examining the distribution of character lengths, patterns may emerge that could be indicative of specific writing styles or contexts within the dataset. In our dataset the majority of the texts were between 0 to 1000 characters.

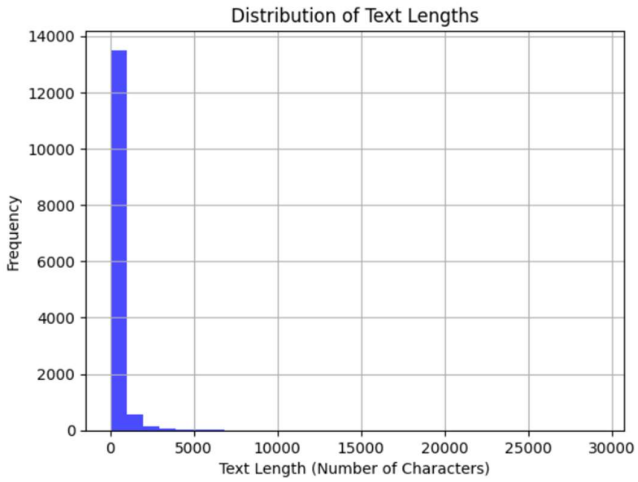


Fig 09: Histogram of text length frequency

3.2.2 Text Length Analysis (Number of words in each text)

We computed the number of words in each text entry to examine the word count distribution. This allows us to understand the dataset's textual makeup. This also allows us to identify texts that are unusually brief or verbose. In our dataset, most texts were between 0 to 250 words..

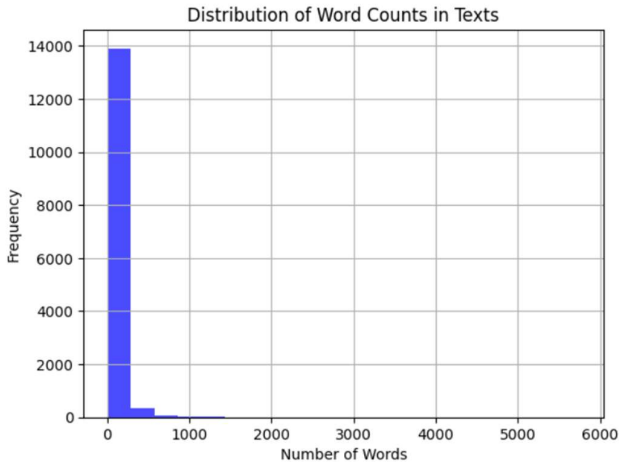


Fig 10: Histogram of word count frequency

3.2.3 Number of words in each class

Analysing the distribution of word counts across different classes or categories within our dataset can reveal variations in text length among these classes. This information assists in understanding potential differences in writing styles, topics, or contexts across various classes. Observing disparities in word counts among classes may indicate distinctive linguistic patterns or thematic variations that could be significant in subsequent analyses.

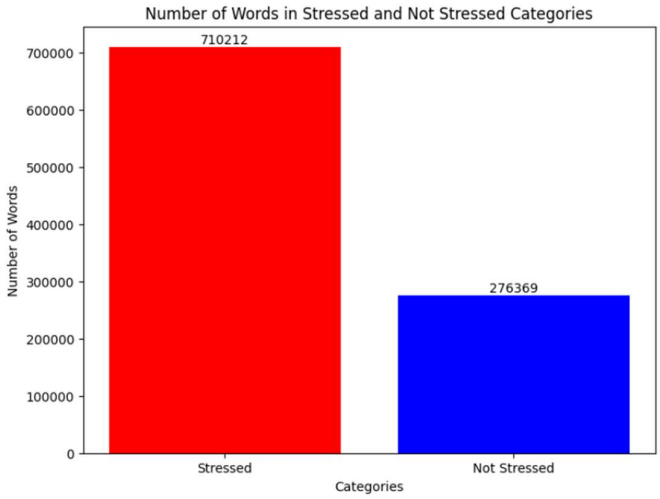


Fig 11: Bar Chart of word count for each label

3.2.4 Average length of words

The average length of the words in the dataset was 5.

This analysis computes the average length of words present in the dataset. The average word length can offer insights into the complexity of language used within the texts. Understanding the vocabulary richness and average word length aids in discerning different writing styles or domains within the

dataset. Longer average word lengths might imply technical or formal language, while shorter lengths could indicate simpler or more conversational language.

3.2.5 Frequency of Stopwords and Common Words

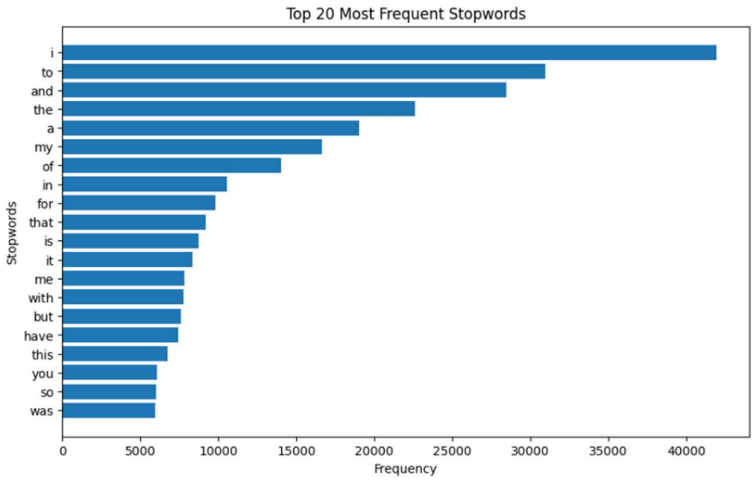


Fig 12: Stopword Frequency

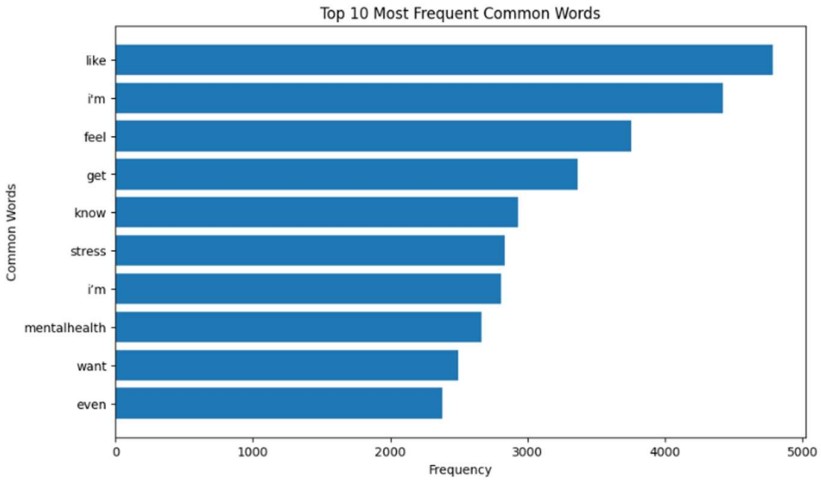


Fig 13: Common Word Frequency

3.2.6 Word Frequency Analysis - Word Clouds

The generation of word clouds visually represents the most frequently occurring words in the dataset. These visualisations offer an intuitive snapshot of dominant terms, facilitating the identification of key words or terms prevalent within the text corpus. Larger words in the word cloud indicate higher frequency, potentially signifying their significance within the dataset.

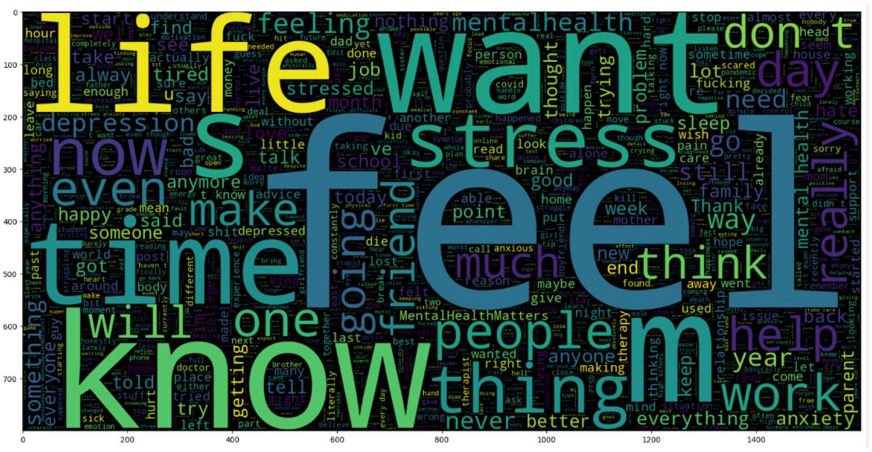


Fig 14: Word Cloud for “Stressed”

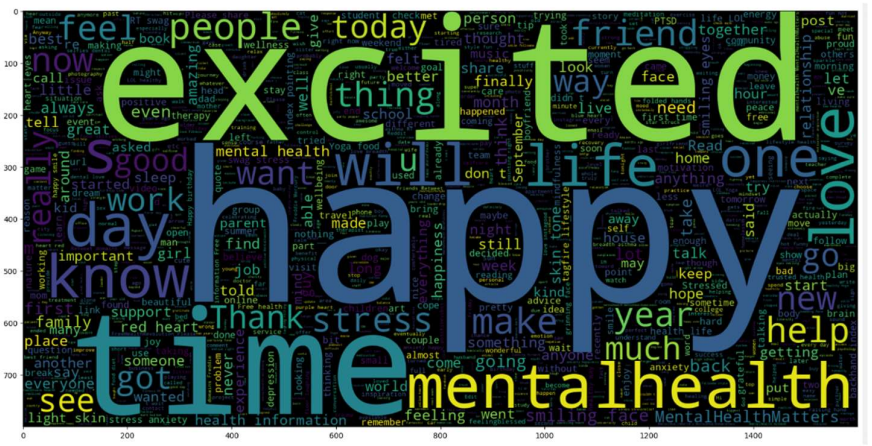


Fig 15: Word Cloud for “Not Stressed”

3.2.7 N-gram Analysis

N-gram analysis investigates sequences of 'n' contiguous items, usually words, within the text data. This technique captures sequential relationships between words that might not be evident in single-word analysis. It provides deeper linguistic insights and aids in understanding contextual patterns within the text corpus. Subcategories such as Unigram, Bigram, and Trigram represent different 'n' values, highlighting single words, pairs, or triplets of words, respectively. These analyses support feature extraction for predictive models and offer valuable information for language-based tasks.

3.2.7.1 Unigram

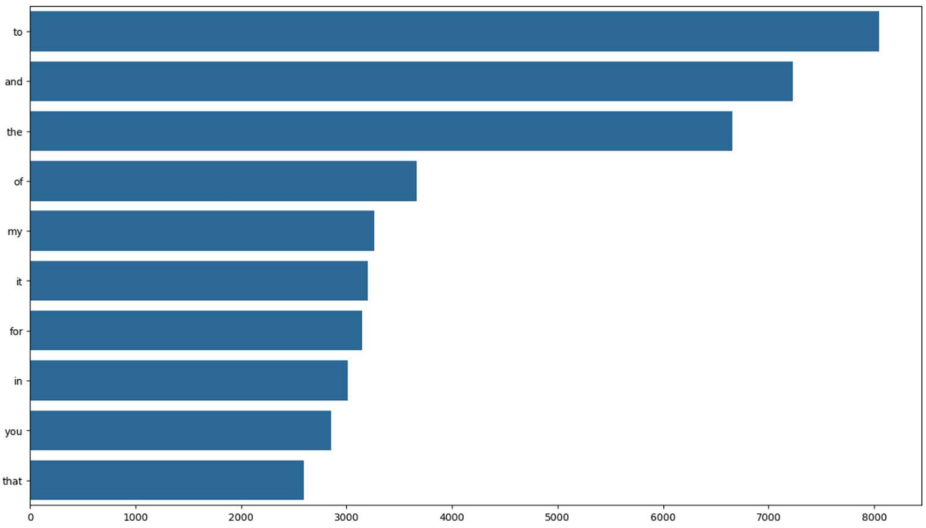


Fig 16: Unigram for Not Stressed

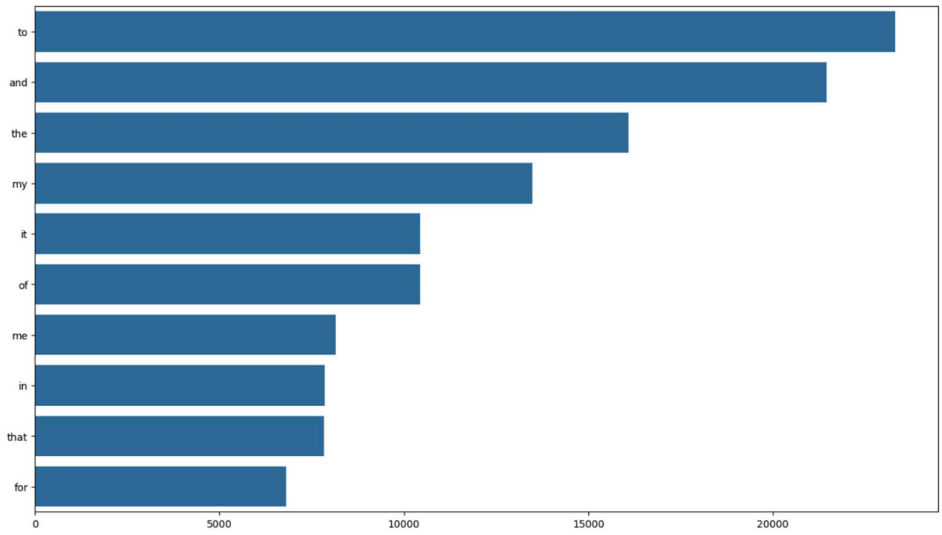


Fig 17: Unigram for Stressed

3.2.7.2 Bigram

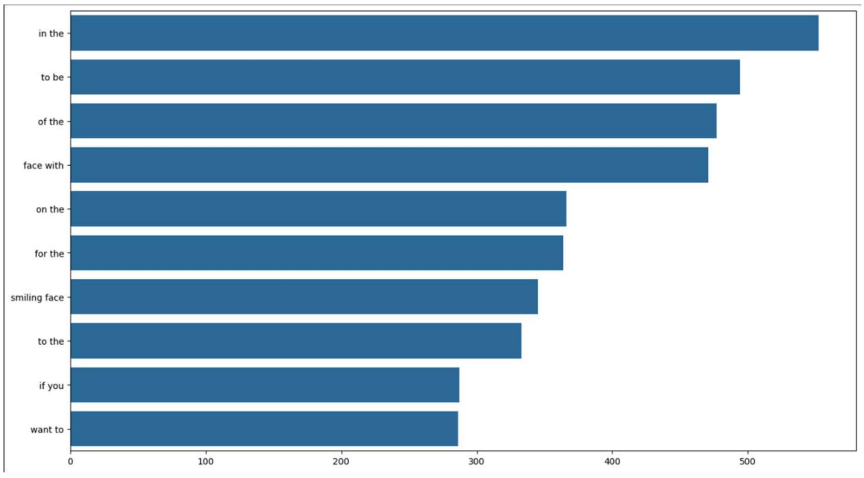


Fig 18: Bigram for Not Stressed

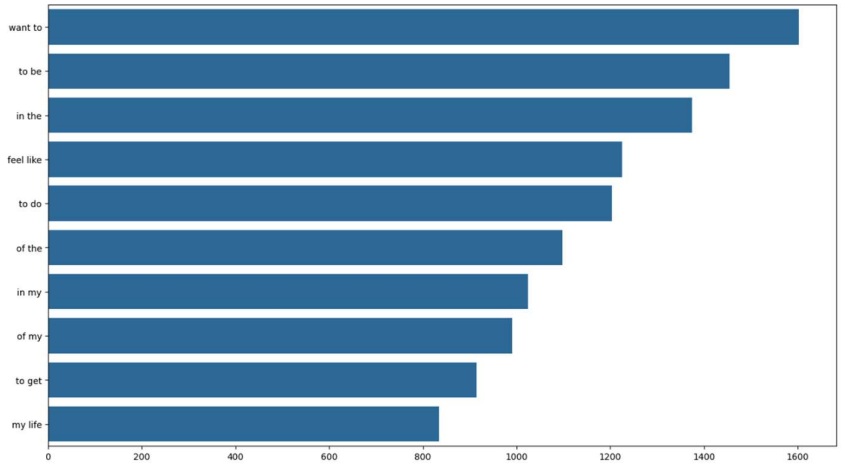


Fig 19: Bigram for Stressed

3.2.7.3 Trigram

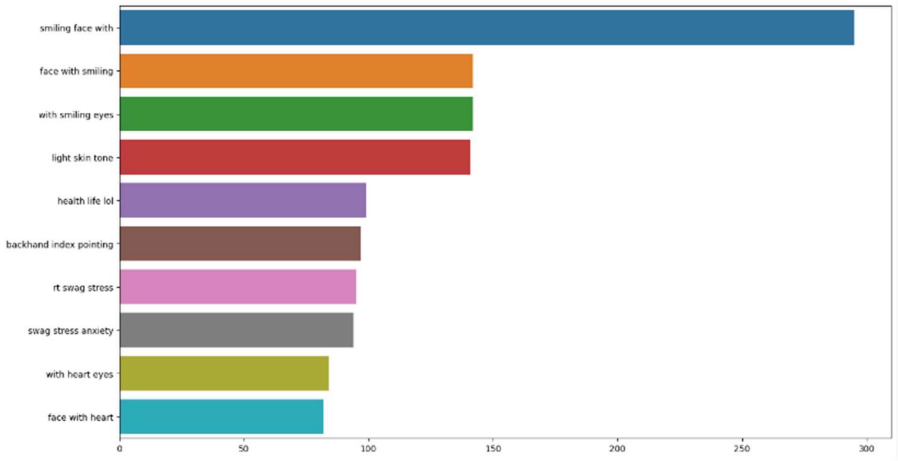


Fig 20: Trigram for Not Stressed

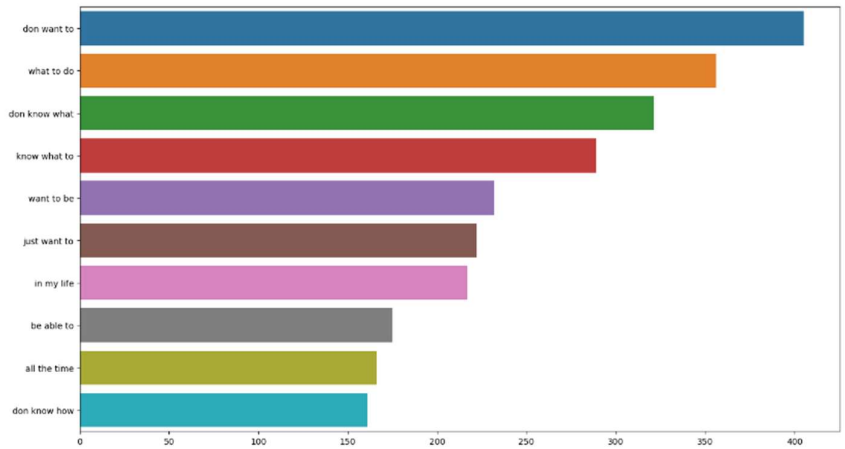


Fig 21: Trigram for Stressed

4. Methodology

In this section, we will discuss the steps we took in completing the task of the binary classification of stress and not stress. In this project, we mainly explored two different methodologies for stress classification: classic machine learning models and deep learning neural networks. The effectiveness of a model would mean nothing if the quality of the data is low. We will now discuss the steps we took in achieving clean data (that is interpretable by machines) for model training and how we conducted our hypothesis testing.

4.1 Data Preprocessing

4.1.1 Text Cleaning

Our initial dataset are actual Reddit posts and Twitter (X) tweets, which contained text in both uppercase letters, lowercase letters and punctuation. This format is not appropriate for model training, and hence we conducted text cleaning by converting all text to lowercase, and removing punctuation from the text.

After the first round of cleaning, the team looked at the data and realised that the data still contained many special symbols (due to the fact that it is possible for users to use special characters/symbols or even emoticons in their posts). Hence, we filtered out all non alphabetical characters.

List of all symbols removed:

['!', '"', '#', '$', '%', '&', "'", '(', ')', '*', '+', ',', '-', '.', '/', ':', ';', '<', '=', '>', '?', '@', '[', '\\', ']', '^', '_', '`', '{', '|', '}', '~', '”', 'ଡ', '𝕜', 'ウ', '\u2069', 'ے', '症', 'ы', '𝐚', 'ё', '𝟒', '𝐨', '𝐡', '𝓎', 'ⅴ', '失', '…', 'س', 'з', 'ɒ', '\u200a', '🎉', '🐰', 'ь', 'ム', '𝐬', 'ा', '𝐜', '—', 'و', '𝐈', '🅔', 'ᴗ', '𝐛', 'í', '°', '스', 'w', '처', '\u200d', '𝓇', 'ᵕ', '̩', '\u200b', 'т', '❤', 'ى', '𝕦', '❝', 'ح', 'ए', '𝐨', '𝐒', 'г', 'ن', '👩', '●', '𝐡', 'c', '\U000d0062', '𝐃', '“', 'ظ', '̀', '𝑔', '₹', '🅢', '𝐌', '출', '𝐒', '🅑', '요', '𝐟', 'ö', '̆', '𝒽', 'ス', '𝕒', 'ᴀ', '𝒻', '🅓', '𝐤', '𝐿', '𝐩', '̀', '•', 'م', '̆', '𝐄', '・', 'ッ', 'ी', '̇', 'ī', '𝕠', 'ハ', 'ے', '𝐅', '🅝', '出', 'ئ', '♤', '𝐘', 'ج', '\ufeff', 'ज', '洲', 'イ', '\u2063', '<', 'e', 'ه', '𝐝', '□', '『', '‣', 'ب', '랑', '🅤', 's', '😔', '피', '💕', '́', 'ा', '𝐬', 'л', 'उ', '🅖', 'ﮦ', '½', '’', '𝐯', 'ᴇ', 'a', '𝕡', '𝐋', '𝐩', '🎵', '̊', 'र', '𝐄', 'x', 'п', '𝓀', '𝕪', 'ボ', '𝓂', '사', 'ネ', '𝐮', '𝕞', 'ت', '𝕨', 'ä', 'ñ', '𝐐', '²', '𝕚', 'ラ', 'ド', '\U000d007f', '🅐', '𝔸', 'Φ', 'α', '✓', '↓', '𝐢', 'к', '′', 'ر', '민', ' ', '\u2066', '𝐓', '€', '→', '𝟕', 'ʟ', 'в', '：', '‘', '𝐠', '𝕣', '£', 'ع', '☆', '💸', '𝐱', '𝐁', 'ك', '𝐭', 'ร', '재', '讓', '𝐧', 'ट', '𝐰', 'ə', '𝐃', 'н', 'д', '𝕤', '"', '𝐠', '🅘', 'B', 'ि', 'त', '\u200e', '𝕖', '♡', '𝕥', '🅨', '𝓋', 'ゲ', '„', '\xad', '😳', '赋', 'ش', 'á', '𝐰', '𝐏', '𝐫', '𝐋', 'ف', '。', 'ا', 'ꜰ', '𝐞', 'ں', 'フ', '>', '露', '\U000d0067', '𝟓', 'オ', '𝕔', '𝐜', '『', '😇', 'p', '»', '𝓡', '𝕧', '𝑎', '𝐭', '◡', '𝕞', '”', '𝐑', '𝑜', '３', 'T', '', '़', '〜', '𝐁', 'é', 'ᶢ', 'स', '➡', '。', 'ï', '🅒', '–', '本', '𝐟', '𝕙', 'à', 'Ц', '＃', '𝐑', '𝕘', '𝒾', 'ث', 'ᴄ', 'è', '𝑐', 'ช', 'ي', '러', '-', 'ช', '𝐇', 'ʀ', 'چ', '‘', '一', '𝐥', '𝐯', '𝐍', '\u202f', '𝕗', '𝓉', '🅣', '\U000d006e', 'ピ', '𝐀', '𝕓', '우', '日', '𝐖', '🅞', '\u2060', 'м', '—', '𝐥', '𝐦', '𝓜', '𝐀', '𝓃', 'ﷺ', 'ü', 'ء', '智', '𝐫', '𝐞', 'å', '😨', '𝐮', '𝕀', '𝐢', 'خ', 'ר', 'т', 'ک', 'ч', '°', 'ʜ', '"', 'O', '\xa0', '·', '플', 'й', 'ص', 'ہ', '𝐥', '하', '𝐘', 'ᴏ', '➤', 'श', '«', '𝐲', 'и', '𝑒', 'ר', '🅡', 'ツ', '해', 'É', 'o', '𝐔', 'ط', '𝐚', '𝕕', '一', '¯', '𝐝', '𝐓', '—', '،', 'ч', '⚘', '𝐤', '🎓', '亚', '𝐧', '𝐖', 'и', 'क', 'म', 'ﬁ', '\U000d0065', '🅗', '', 'थ', 'ℍ', 'ل']

4.1.2 Stop Words Removal

Stop words are words that do not tell us much meaningful information about our data. They are words that exist in our language to help humans understand each other better. For example, some stop words that are used in this paragraph are: “are”, “that”, “do”, “not”, “us” etc. The reason why we want to remove stop words is to remove the computational complexity and reduce the size of the dataset for more efficient model training. (Khanna, 2021)

	Categorization object	Body object	La...	Cleaned Text object
0	Envy to other is s...	Im from developingcountry, Indonesia , and for n...	1	im developingcountry indonesia temporary work ov...
1	Nothin outta the o...	Um hello ....well many can relate im sure. After t...	1	um hello well many relate im sure today im convince...
2	Almost 49 and the...	I've been diagnosed severe bi polar where you n...	1	ive diagnosed severe bi polar longer even get good ...
3	I'm happy again	After my closest friend left me in April, I have fin...	0	closest friend left april finally let go realized much n...
4	Is it possible to re...	I am only 15, and yet I feel my life is already over...	1	15 yet feel life already pit emptiness stomach dont s...

Fig 22: Effect on text after removal of stop words

4.1.3 Text Lemmatization

Now that we have removed words that will not have a meaningful impact on our results, we are left with a long list of words that tell us useful details about our data. The next step is to shorten the length of the words, to just the parts that help us identify its meaning.

Two methods come to mind, stemming and lemmatization. Stemming is where you remove the ending parts of the word, to obtain the stem(root word that tells us the meaning). The second method, lemmatization is where we retain semantic knowledge. For example, given the word “better”, stemming would give us “bet” or “bett” whereas lemmatization can reduce it to “good”, which is closer to actual meaning. (Srinidhi, 2023)

We chose to do lemmatization because we valued the improvement in accuracy in helping our model understand the words better. While we noted that this might increase the time taken and computational power needed, we acknowledged that given the size of our dataset, it is worth it for this improvement in accuracy.

Body	Cleaned Text	Cleaned Text with N lemmatization	Cleaned Text with V lemmatization	Cleaned Text with A lemmatization
Im from developingcountry, Indonesia , and for now i temporary work overseas for 3 years contract, it's a hard labor job, and stressful. Next year my contract is finish. But, during my stay here, because of job, and my social life, my depression got worse, and i envy this developed country. Why this country is so good. I can afford anything i want here. Why can't we just have equality in currency exchange? I I need to work 15-20 years in big company in jakarta(our capital city) , just to get equal amount of saving money from what i got from 3 years working here. Yes, that's right, it's saving money, not spending money. And yes, im going to be a rich person in such young age If I think about it, this society is sick, the gap of un equality beetwen developing vs developed country, or the poor vs the rich is too big right now. Sorry if i look like an evil person , but because of this,I almost wish for war to happen, or this world to end, and got reseted. So everyone can have another chance to gain equality, the poor can finally have a chance to have a better life. Because if this world order stay this way, it will eventually collapse on itself. Soon.	im developingcountry indonesia temporary work overseas 3 years contract hard labor job stressful next year contract finish stay job social life depression got worse envy developed country country good afford anything want cant equality currency exchange need work 1520 years big company jakartaour capital city get equal amount saving money got 3 years working yes thats right saving money spending money yes im going rich person young age think society sick gap un equality beetwen developing vs developed country poor vs rich big right sorry look like evil person thisi almost wish war happen world end got reseted everyone another chance gain equality poor finally chance better life world order stay way eventually collapse soon	im developingcountr y indonesia temporary work overseas 3 year contract hard labor job stressful next year contract finish stay job social life depression got worse envy developed country country good afford anything want cant equality currency exchange need work 1520 year big company jakartaour capital city get equal amount saving money got 3 year working yes thats right saving money spending money yes im going rich person young age think society sick gap un equality beetwen developing v developed country poor v rich big right sorry look like evil person thisi almost wish war happen world end got reseted everyone another chance gain equality poor finally chance better life world order stay way eventually collapse soon	im developingcount ry indonesia temporary work overseas 3 years contract hard labor job stressful next year contract finish stay job social life depression get worse envy develop country country good afford anything want cant equality currency exchange need work 1520 years big company jakartaour capital city get equal amount save money get 3 years work yes thats right save money spend money yes im go rich person young age think society sick gap un equality beetwen develop vs develop country poor vs rich big right sorry look like evil person thisi almost wish war happen world end get reset everyone another chance gain equality poor finally chance better life world order stay way eventually collapse soon	im developingcountr y indonesia temporary work overseas 3 years contract hard labor job stressful next year contract finish stay job social life depression got bad envy developed country country good afford anything want cant equality currency exchange need work 1520 years big company jakartaour capital city get equal amount saving money got 3 years working yes thats right saving money spending money yes im going rich person young age think society sick gap un equality beetwen developing vs developed country poor vs rich big right sorry look like evil person thisi almost wish war happen world end got reseted everyone another chance gain equality poor finally chance good life world order stay way eventually collapse soon

Figure 23: Comparison of original text against different lemmatizations

4.2 Post Data Preprocessing Analysis

After doing data preprocessing, we can clearly see that the Trigram for stressed clearly reflects the inherent words which makes it easier to infer that the body text is labelled 1 (stressed). The changes are reflected in Figure 22 below.

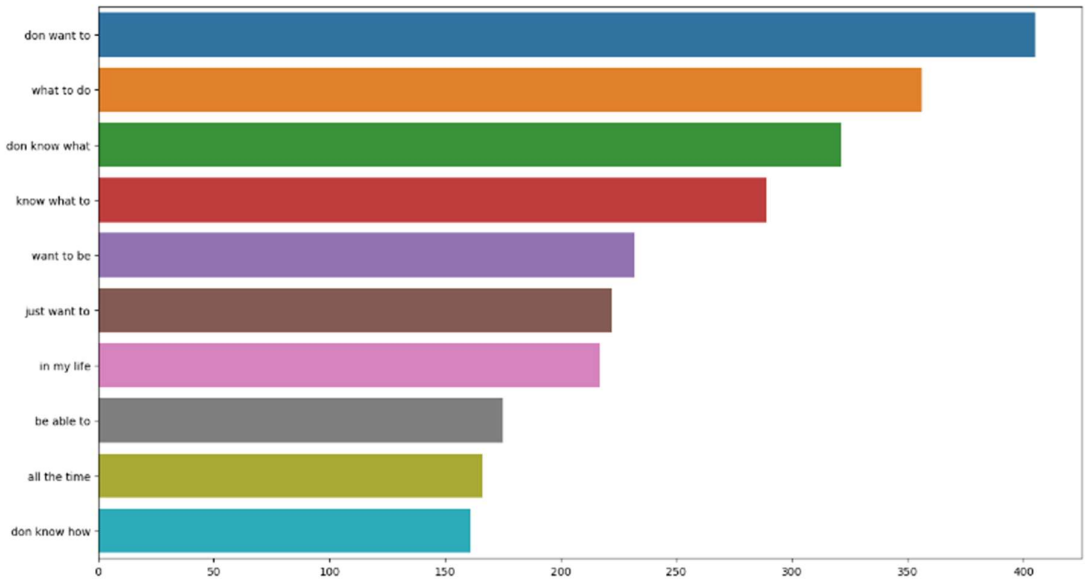


Figure 24: Trigram of stress before data preprocessing

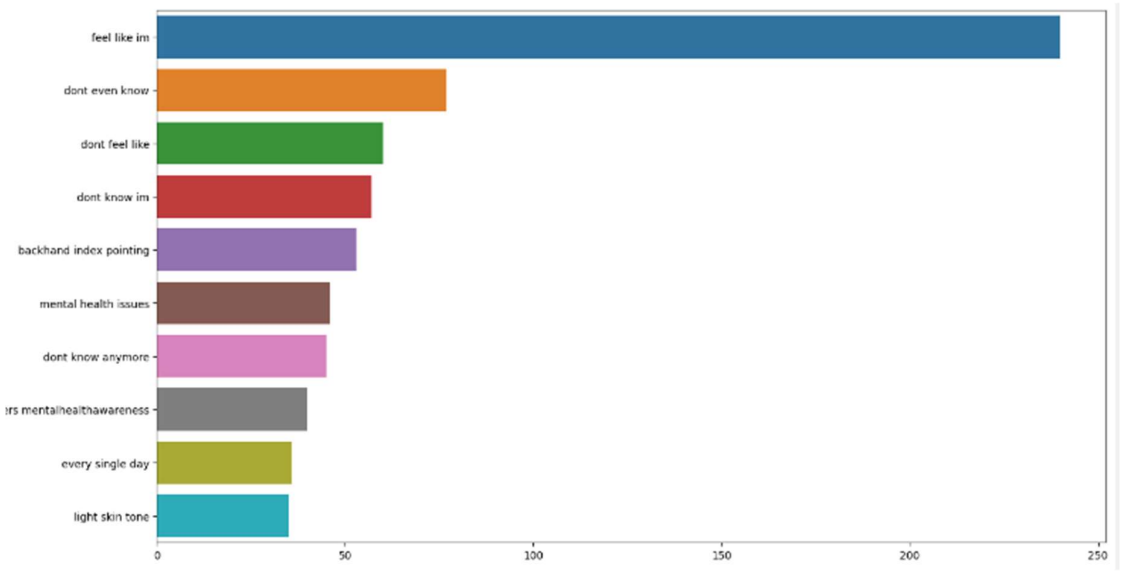


Figure 25: Trigram of stress after data preprocessing

4.3 Hypotheses

4.3.1 Hypothesis 1: How does different text lemmatization affect the model results?

There are different parts of speech (pos) to lemmatize by

1. Noun (pos = 'n'):
    a. "cats" ->"cat"
    b. "dogs" -> "dog"
    c. "books" -> "book"

2. Verb (pos = 'v')
    a. "running" -> "run”
    b. "eating" -> "eat"
    c. "swimming" -> "swim"
2. Adjectives (pos = "a")
    a. better -> "good"
    b. "happier" -> 'happy'
    c. "brighter" -> "bright"

4.3.2 Hypothesis 2: How does different text extraction methods affect the results?

Our second hypothesis is to determine how employing different text extraction methods influences the results. We decided to use the following text embedding method to see how they affect the results. Each of the techniques aims to represent words or documents as dense, fixed-size vectors in a continuous vector space, capturing semantic and contextual information.

4.3.2.1 Text Extraction Methods used

4.3.2.1.1 Global Vectors for Word Representation (GloVe)

Description: GloVe is designed to capture the semantic relationships and meaning of words based on their co-occurrence statistics in large text corpora (Pennington et al., 2014) . Its key advantage is its ability to capture word semantics and relationships through the distributional information in large text corpora. The resulting word vectors can be used as features in NLP

Pros: Dimensionality reduction: The feature matrix has a lower dimensionality compared to direct document vectors.
Simplicity: Averages can capture the overall meaning of the text while reducing the feature space.

Cons: Loss of word order: Averaging word embeddings doesn't capture word order information, which might be important in some cases.
May not capture all nuances: Averaging can simplify the representation and might not capture nuanced differences in text.

Feature Transformation: nil
Explanation:
There isn't a need for additional feature transformation steps like TF-IDF, PCA, or LDA because the word embeddings themselves already capture a dense, distributed representation of words. GloVe word embeddings inherently contain semantic and contextual information.

In the case of GloVe embeddings, each word is represented as a high-dimensional vector, and these vectors capture relationships between words based on their co-occurrence statistics in the training corpus. This means that words with similar meanings or contexts will have similar vector representations.

Therefore, we can directly use GloVe word embeddings as features for your text classification model without the need for feature transformation steps

Pre-Trained GloVe Embedding Model

The pre-trained GLoVe embedding we used was trained on Twitter (X) datasets. We decided that this would be most appropriate as our project involves social media posts and comments.

Link to GloVe Embeddings: https://nlp.stanford.edu/projects/glove/

We had 2 different approaches to use GloVe embedding

- **Approach 1:** Average Word Embeddings with GloVe
    - **Explanation**: The approach calculates the average word embeddings for each text sequence. Each document is represented as an average of word vectors. We decided to explore how well the individual words can help us to predict whether a post will be classified as stressed.
- **Approach 2:** Direct GloVe Document Vectors
    - **Explanation**: The approach directly used GloVe embeddings for feature extraction. Each document or text sequence was represented as a single vector. Our thought process is that this version should perform better because it retains the semantic information of the entire document.

Furthermore, we experimented with different vector sizes as GloVe embeddings are pre trained with specific parameters (window,min_count,vector_size) that can't be changed. Hence, we ran our model with different vector sizes of 50, 100 and 200 to compare which vector size would be most appropriate.

4.3.2.1.2 Word2Vec

Description: Word2Vec is a powerful natural language processing technique for capturing semantic relationships and word context within text data. It generates word embeddings, which are dense vector representations of words, by considering the co-occurrence patterns of words in a large corpus of text (Dutta, 2023). These word embeddings encode semantic information and can be leveraged as feature vectors for various text processing tasks.

While we thought it would have been more interesting to train our own Word2Vec model, it was too computationally intensive for our CPU, and we decided to capitalise on Google's pre-trained vectors that have been trained on about 100 billion words from the Google News dataset. The model has 300-dimensional vectors for about 3 million words and phrases.

Link to Pre-trained Word2Vec Embeddings: https://code.google.com/archive/p/word2vec/

Pros: Semantic understanding: Captures semantic relationships between words, and words with similar meanings are represented as vectors that are close in the vector space. This enhances the ability of Word2Vec to capture the meaning of words and phrases effectively.

Contextual information: Considers the context in which words appear, enabling it to distinguish between different uses of the same word based on the neighbouring words. This contextual understanding is important for tasks like word analogy and text similarity.

Smaller feature space: Unlike traditional bag-of-words models, Word2Vec generates relatively low-dimensional vectors, making it computationally efficient and well-suited for downstream machine learning models.

Cons: Data dependency - The effectiveness is highly dependent on the quality and quantity of the training data. It requires a substantially large amount of text data to capture meaningful word relationships, which may not be available in certain cases.

Out-of-vocabulary words: Word2Vec may have difficulty with out-of-vocabulary words if they are not seen during training. Handling such words requires additional techniques or using subword embeddings like FastText.

Feature Transformation: None
Explanation: Word2Vec embeddings provide a rich and semantically meaningful representation of words without the need for additional feature transformation. These embeddings are ready for use in various natural language processing tasks, including text classification, sentiment analysis, and document similarity, where capturing the meaning and context of words is crucial for accurate results.

4.3.2.1.3 FastText

Description: FastText is an effective method for capturing the semantics and context of words through subword information. It excels in handling out-of-vocabulary words and languages with rich morphology. FastText generates word embeddings based on character-level n-grams and their co-occurrence statistics in text data (Krithika, 2023). These embeddings are valuable features for natural language processing tasks.

Pros: Subword information: FastText can handle out-of-vocabulary words and complex word forms, making it suitable for diverse languages.
Contextual understanding: FastText embeddings capture the semantics and context of words, offering a rich feature representation for text data.

Cons: Larger feature space: Each document is represented as a high-dimensional vector, potentially leading to a higher dimensionality.

Feature Transformation: nil
Explanation: No additional feature transformation steps are necessary. FastText word embeddings inherently capture subword information, semantics, and contextual understanding. These embeddings offer a dense, distributed representation of words, enabling direct use as features for text classification without the need for further feature transformation.

4.3.2.1.4 Doc2Vec

Description: Doc2Vec is an algorithm that extends the concept of word embeddings to entire documents. It generates fixed-length vector representations for variable-length pieces of texts, such as sentences, paragraphs, or documents (Khare, April 2). Doc2Vec aims to capture the semantic meaning of documents by learning distributed representations that encapsulate the context of words in the document.

Pros: Document-level embeddings: Doc2Vec generates embeddings for entire documents, allowing it to capture the context and semantics of the entire text, which can be valuable for tasks involving document-level understanding.

Semantic understanding: By creating document vectors that encapsulate the meaning of the entire document, Doc2Vec can support tasks requiring semantic similarity or document classification.

Cons: Complexity and training time: Training Doc2Vec models can be computationally expensive, especially with large datasets and higher vector dimensions.

Interpretability: While Doc2Vec creates embeddings for documents, interpreting these embeddings might not be as straightforward as interpreting word-level embeddings.

Feature Transformation: -

Explanation: Doc2Vec, similar to Word2Vec for words, generates fixed-length vector representations for entire documents. These document embeddings capture the semantic meaning and context of documents, allowing direct use as features for various natural language processing tasks, including document classification, retrieval, or similarity calculations.

4.3.2.1.5 Embeddings from Language Models (ELMo)

Description: ELMo (Embeddings from Language Models) is a deep contextualised word representation model developed by researchers at the Allen Institute for Artificial Intelligence. Unlike traditional word embeddings, which assign a fixed vector to each word, ELMo generates contextualised word representations that capture various aspects of word meaning and usage based on the specific context in which the word appears (Joshi, 2022).

Pros: ELMo captures contextual information, allowing for a more nuanced understanding of word meanings based on their surrounding context, which can be beneficial for various downstream NLP tasks.

Cons: ELMo models can be computationally expensive and may require significant computational resources, making them less feasible for resource-constrained environments. ELMo embeddings also heavily rely on large and diverse datasets for training, which might limit their effectiveness when working with domain-specific or small-scale datasets.

Feature Transformation: nil

Explanation: Similarly to GloVe word embeddings, ELMo also does not require additional feature transformation steps like TF-IDF, PCA or LDA because the word embeddings are context-sensitive word representations by considering the entire input sentence. It uses a deep, bidirectional language model to compute word embeddings that capture both syntax and semantics, considering the surrounding context.

4.3.3 Hypothesis 3: Which model performs best for the classification of text?

Our third hypothesis is to determine which model is best suited for our project whereby we aim to classify whether the person is stressed or not stressed from the text.

We trained classic machine learning models to classify the texts in the dataset. Specifically, we used logistic regression, multinomial Naïve Bayes, Random Forest, XGBoost and linear support vector machines (SVMs), which are widely used for text classification tasks. In addition to the classic machine learning models, we also explored the use of deep learning neural networks for text classification.

4.3.3.1 Models Used

4.3.3.1.1 Logistics Regression

Logistic regression is a statistical technique used for binary classification tasks in text analysis. It models the probability of a text belonging to a specific category by employing the sigmoid function, which produces a smooth, S-shaped curve.

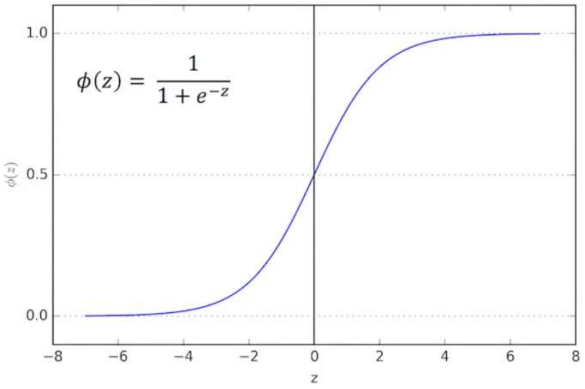
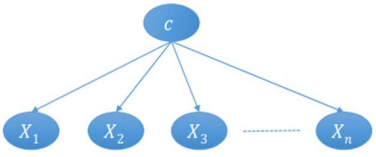


Fig 27: Graphical representation of Sigmoid Function (Piduguralla, 2023)

4.3.3.1.2 Multinomial Naive Bayes

Multinomial Naive Bayes is a variant of the Naive Bayes algorithm, which is based on Bayes' theorem. It is a generative probabilistic model which gives conditional probability based on prior probability and likelihood to obtain posterior probability. The predicted label with the highest posterior probability is the class of the dataset. In text classification, it models the distribution of word occurrences in documents. It assumes that each term (word) is drawn from a multinomial distribution.



Conditional Independence assumption: variables X_i are independent of each other given variable c.

- $X_i \perp X_j \mid c$

Fig 28: Graphical representation of Naive Bayes

4.3.3.1.3 Random Forest

A Random Forest Classifier is an ensemble machine learning model that is primarily used for classification tasks, although it can also be adapted for regression. It's part of the ensemble learning family, which combines the predictions of multiple individual models to improve overall predictive accuracy and reduce overfitting. Random Forests are known for their robustness, ease of use, and ability to handle a wide range of data types.

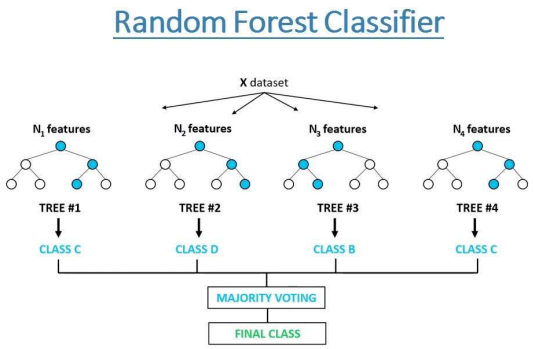


Fig 29: Graphical representation of the Random Forest Classifier (Khushaktov, 2023)

4.3.3.1.4 XGBoost

eXtreme Gradient Boosting or XGBoost for short, is a popular algorithm that has risen to fame due to its high performance at Kaggle competitions. We were also excited to see how it would fare for our dataset, as compared to the more complex models like Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks. XGBoost uses decision trees as the base estimator, but the way it builds trees is different from typical Gradient Boosting. XGBoost has its own method of building trees where the Similarity Score and Gain will decide which split is the best at the current stage.

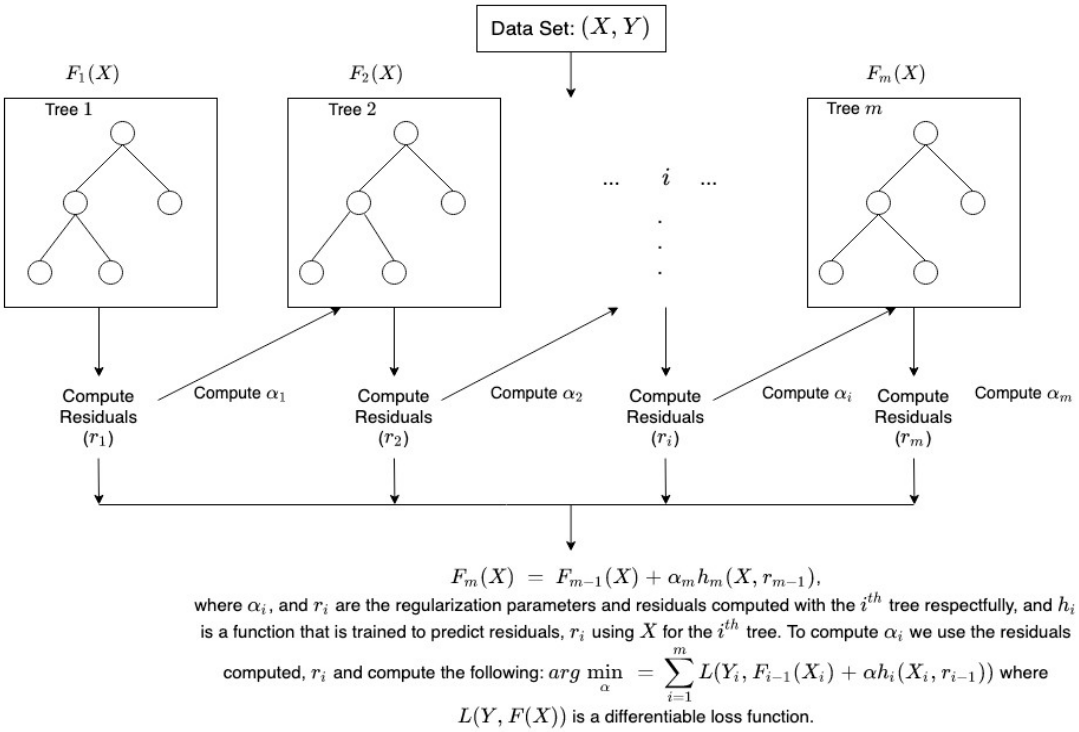


Fig 30: Graphical representation of how Gradient Boosting works (Amazon Web Services, 2023)

4.3.3.1.5 Support Vector Machine (SVM)

Support Vector Machine (SVM) is a supervised machine learning algorithm that is used for classification and regression tasks. It is particularly popular for classification problems. SVM aims to find a hyperplane that best separates data into different classes in a way that maximizes the margin between the classes.

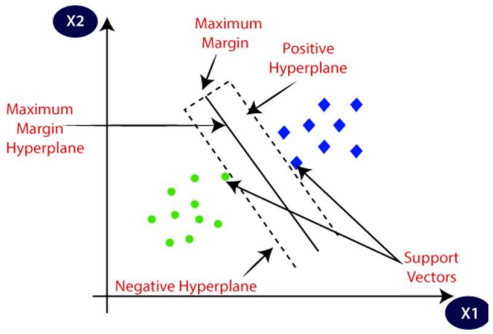


Fig 31: Graphical representation of how SVM works (Wikipedia, 2023)

4.3.3.1.6 Recurrent Neural Network (RNN)

Recurrent Neural Networks (RNNs) are a type of neural network architecture designed for processing sequential data and time series information. Unlike traditional feedforward neural networks, RNNs have

connections that loop back on themselves, allowing them to maintain a memory of previous inputs. This looped structure enables RNNs to capture dependencies and patterns within sequences.

For activation function, tanh is often preferred in RNNs due to its ability to handle both positive and negative information, which can be beneficial for capturing long-term dependencies. tanh squashes input values to the range [-1, 1], which can help in mitigating the vanishing gradient problem by centering the data around zero.

After hypertuning, we determined that the optimal configuration for the RNN model involved using the hyperbolic tangent (tanh) activation function with 100 units. Following this, a Dense layer with a sigmoid activation function was added, and the model was compiled using the Adam optimizer with a learning rate of 0.0001. Additionally, 10 epochs and a batch size of 64 were identified as the parameters that yielded the best performance for the RNN architecture. This configuration enables the RNN to effectively learn and make predictions on sequential and time-dependent datasets.

4.3.3.1.7 Long Short Term Memory (LSTM)

LSTM is a type of recurrent neural network (RNN) architecture that is well-suited for learning sequences and time series data. LSTM networks can maintain information in memory for long periods. This capability allows them to effectively learn from and make predictions based on sequences of data. After hypertuning, we found the best parameters to be ReLU activation function with 100 units. This was followed by a Dense layer with sigmoid activation function and finally a compilation layer using Adam optimizer with a learning rate of 0.01. We also found 10 epochs and 64 batch sizes to give us the best performance.

4.4 Workflow

4.4.1 Workflow for ML Models

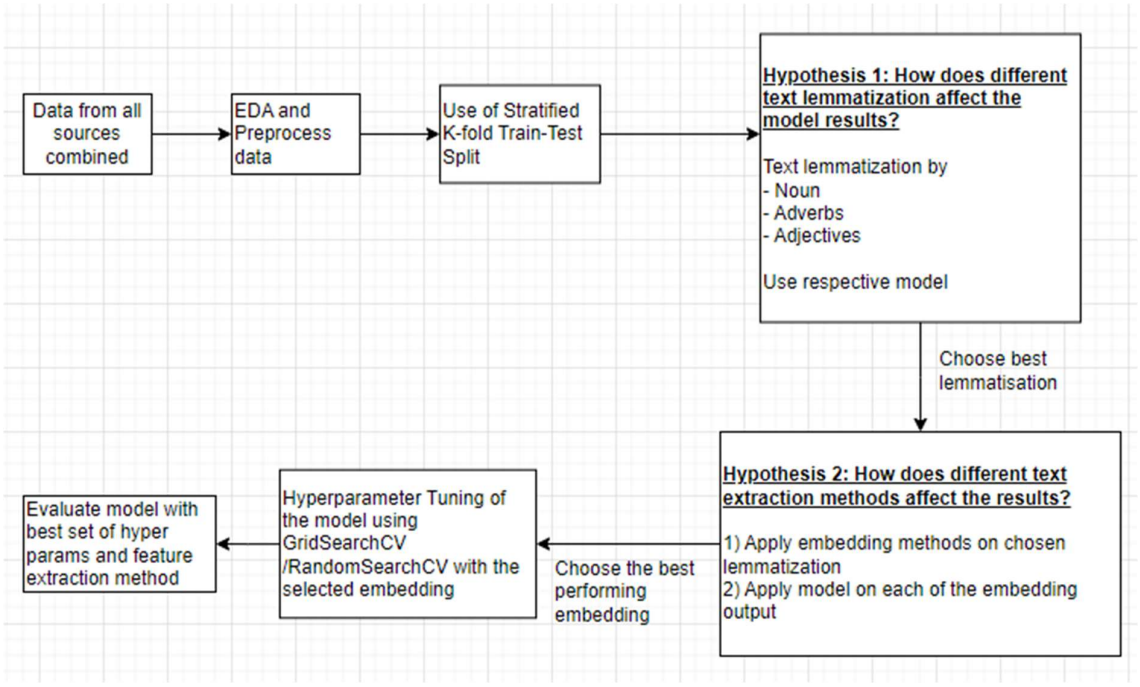


Fig 32: Workflow for ML Model

A basic workflow that we followed for the classic ML models was that we first combined the 2 dataset we had. Next we did EDA and Preprocessed the dataset. Then we used a stratified k fold train test split for the models.

We first carried out hypothesis 1 to find the best text lemmatization method. We used the best text lemmatization column to then carry out hypothesis 2. We first applied the embedding methods on the chosen text lemmatized column. Thereafter we applied the model on the embedding output, and determined the best performing embedding method. Lastly we hypertuned the model using gridsearch/random search cv before evaluating the model with the best params and text extraction method.

4.4.2 Workflow for Deep Learning Models

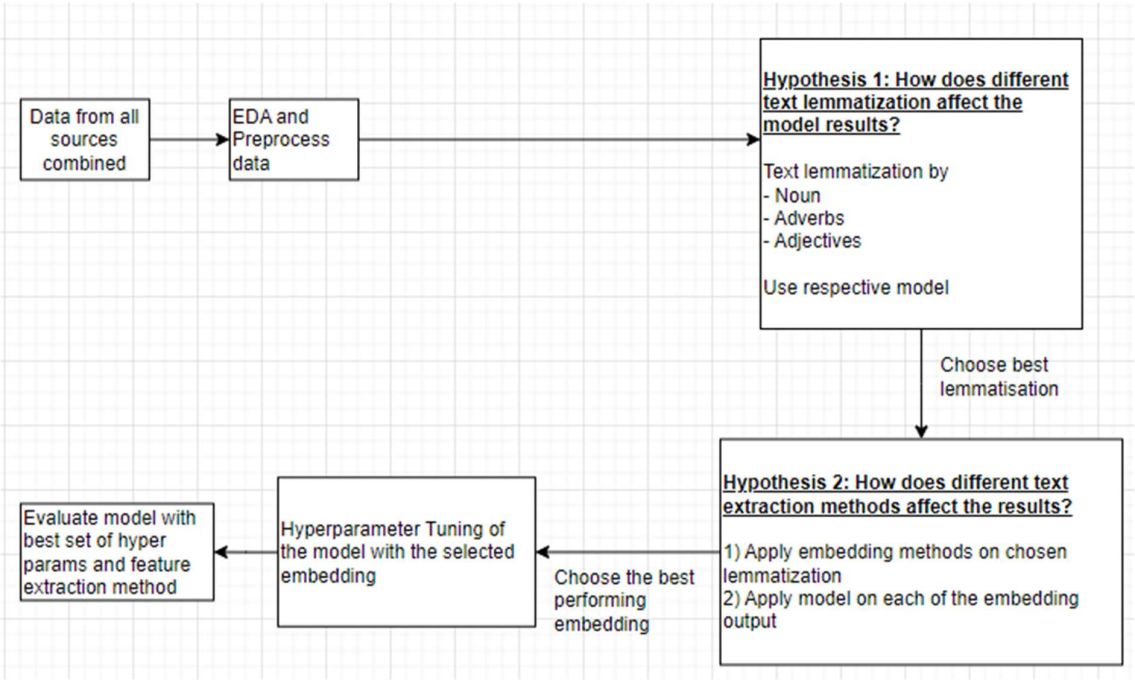


Fig 33: Workflow for Deep Learning Model

Deep learning models follow a similar workflow, except the use of stratified K-fold train test split

4.5 Evaluations Metrics

4.5.1 Sensitivity / Recall

Formula: TP / (TP + FN)

In the context of predicting stress, it is a measure of how often the model correctly predicts positive cases out of all the actual positive cases.

Sensitivity is the most important metric as a higher sensitivity would result in more early diagnosis of stress and allow for early intervention. Furthermore, we realised that false negatives, which are people that are stressed but undiagnosed, are a much higher cost as compared to misdiagnosed people who do not have stress. As a result, for this project we have decided to prioritise sensitivity, while trying to strike a balance with a reasonable accuracy.

4.5.2 Accuracy

Formula: (TP +TN) / (TP + TN + FP + FN)

Measure of how accurate the model correctly predicts true cases out of all the cases

4.5.3 Precision

Formula: TP / (TP + FP)

Measure of how precise the model correctly predicts true positive cases out of all the positive cases.

4.5.4 Specificity

Formula: TN / (TN + FP)

In the context of a model predicting stress, it is a measure of how often the model correctly predicts negative cases (i.e. individuals without stress) out of all the actual negative cases.

4.5.5 Classification Error

Formula: (FP+FN)/(TP+TN+FP+FN) = 1- Accuracy

In the context of a model predicting stress, classification error occurs when the model misclassifies an individual as either having stress when they do not (a false positive) or not having stress when they do (a false negative).

4.5.6 F1 Score

Formula: (2 * sensitivity * precision) / (sensitivity + precision)

A combined measure that is derived from Precision (P) and Recall (R) is the F measure. Measures a model's accuracy and computes the number of times a model correctly predicted the entire dataset

4.5.7 Area under ROC curve (receiver operating characteristic curve)

ROC is a probability curve and AUC represents the measure of separability. It tells how much the model is capable of distinguishing between classes. The higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1. The ROC curve is plotted with True Positive Rate (Sensitivity) against the False Positive Rate(1 - Specificity) where TPR is on the y-axis and FPR is on the x-axis. This can be seen in the figure below.

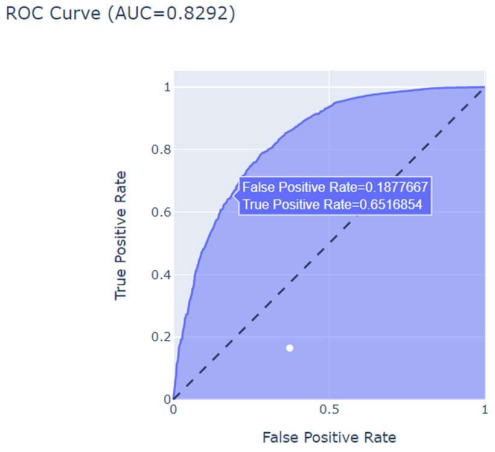


Fig34 . Example ROC Curve to illustrate Area under ROC metric

4.6 Other Tools

4.6.1 Stratified K-Fold Train Test split

Stratification: The dataset is divided into k subsets/folds while ensuring that each fold maintains the same class distribution as the whole dataset. This means that each fold will have a similar ratio of target classes as the original dataset.

Training and Testing: The model is trained on k-1 folds (combined) and tested on the remaining fold. This process will be repeated k times, with each fold used once as the test set.

Performance Metrics: The performance metrics (like accuracy, precision, recall, etc.) are calculated for each fold, and often, the average metric across all folds is taken as the final evaluation metric for the model.

Pros of Stratified k-fold Cross-Validation:

1. **Reduced Bias:** Ensures that each fold has a representative distribution of classes, reducing the risk of bias in model evaluation.

2. **Reliable Performance Estimate:** By averaging the evaluation metric across multiple folds, the performance estimate of the model tends to be more reliable and less sensitive to the partitioning of the data.

3. **Useful for Imbalanced Datasets:** Particularly helpful when dealing with imbalanced datasets, where one class might dominate the data. Ensuring each fold has a similar class distribution helps the model to learn more effectively.

4. **Wider Generalization:** Since the model is trained and tested on multiple subsets of data, it tends to generalise better to new, unseen data.

4.6.2 Hyper Parameter Tuning

Hyperparameter tuning is the process of optimising the parameters of a machine learning model that are not learned during training but rather set before training begins. These parameters, called hyperparameters, significantly impact a model's performance but cannot be directly learned from the data. Through Hyperparameter tuning, we aim to find the **best combination of hyperparameters** that results in the highest model performance.

GridSearchCV and RandomizedSearchCV are the 2 techniques that we used for hyperparameter tuning:

1. **GridSearchCV:** This technique exhaustively searches through a specified grid of hyperparameters. It evaluates the model's performance for **every possible combination** of hyperparameters within the grid and selects the combination that yields the best performance. While it's comprehensive, it can be computationally expensive, especially with larger search spaces.

2. **RandomizedSearchCV:** Unlike GridSearchCV, RandomizedSearchCV randomly samples hyperparameter combinations from specified distributions. It does not exhaustively try all possible combinations but rather narrows down the search space by randomly selecting parameter sets. This approach can be more efficient in high-dimensional spaces and is often preferred when computational resources are limited.

Both methods aim to optimise hyperparameters to enhance a model's performance, but they differ in their search strategies. GridSearchCV is thorough but can be computationally expensive, while RandomizedSearchCV sacrifices exhaustiveness for efficiency, making it suitable for larger search spaces or limited resources. We used GridSearchCV

5. Results and Discussion

Section 5.1 of the report will discuss the results of our binary classification of toxic comments. The models used for this analysis are - Logistic Regression, Multinomial Naive Bayes and Linear Support Vector Machine (SVM). We will explore our findings from the models trained on the undersampled dataset, and similarly, using the same models to predict on the Reddit and Twitter datasets. Finally, comparing their difference in performance.

5.1 Results for Classic Machine Learning Techniques

Limitation: Hyperparameter Tuning Embedding methods are extremely computationally intensive. Due to time and resource constraints for this project, we made the decision to only tune the hyperparameters of the FastText model, which was able to be completed within a reasonable amount of time.

5.1.1 Logistic Regression

5.1.1.1 Results for Hypothesis 1

By using Stratified k-Fold Train Test Split, the model iterated through each fold and calculated the following metrics (accuracy, classification error, sensitivity, precision, specificity, F1 score and AUC) for each fold. After appending each metric to a list and computed the average, the results are as follows:

Text Lemmatization Method	Accuracy	Precision	Specificity	Sensitivity	F1	AUC	Classification Error
Nouns	82.17%	81.44%	69.84%	90.62%	85.78%	90.18%	17.83%
Adjectives	82.12%	81.40%	69.77%	90.58%	85.74%	90.18%	17.88%
Verbs	82.05%	81.63%	70.41%	90.03%	85.62%	90.17%	17.95%
Cleaned Text (Base comparison)	82.10%	81.37%	69.72%	90.58%	85.72%	90.17%	17.90%

Based on the results above, the best performing lemmatization is **Noun Lemmatization.**

5.1.1.2 Results for Hypothesis 2

We chose the best performing feature from hypothesis 1 which is the "**Cleaned Text with N lemmatization**" to apply the different embedding techniques on.

Performed hyperparameter tuning for FastText

Base Model Used: Logistic Regression

Best Parameters: {'vector_size': 200, 'window': 15, 'min_count': 3, 'sg': 1}

By using Stratified k-Fold Train Test Split, the model iterated through each fold and calculated the following metrics (accuracy, classification error, sensitivity, precision, specificity, F1 score and AUC) for each fold. After appending each metric to a list and computed the average, the results are as follows:

Embedding	Accuracy	Precision	Specificity	Sensitivity	F1	AUC	Classification Error
Bag of Words (BoW) with TF-IDF transformation	82.17%	81.44%	69.84%	90.62%	85.78%	90.18%	17.83%
GloVe (version 1) - Vector size 50	78.69%	79.67%	67.92%	86.06%	82.74%	85.95%	21.31%
GloVe (version 1) - Vector size 100	80.40%	81.44%	71.11%	86.77%	84.01%	87.62%	19.60%
GloVe (version 1) - Vector size 200	81.30%	82.11%	72.10%	87.60%	84.76%	88.61%	18.70%
GloVe (version 2)	80.44%	81.45%	71.12%	86.82%	84.05%	87.62%	19.56%
Word2Vec	81.30%	81.61%	70.90%	88.42%	84.88%	88.47%	18.70%
FastText	80.40%	80.65%	69.10%	88.14%	84.23%	87.86%	19.60%
FastText (Hyperparameter Tuned)	80.58%	80.80%	69.34%	88.27%	84.36%	88.31%	19.42%
Doc2Vec	71.43%	71.72%	50.68%	85.64%	78.06%	77.38%	28.57%
Elmo	81.23%	82.89%	74.02%	86.17%	84.49%	88.77%	18.77%

The chosen embedding method is **Bag of Words (BoW) with TF-IDF transformation** as it is the best performing embedding method.

5.1.1.3 Model Hyperparameter Tuning

We hypertuned the Logistic Regression Model with the best performing embedding method to determine the best set of parameters for the Logistic Regression Model via Grid Search CV.

- **Base Model**: Logistic Regression
- **Embedding method**: Bag of Words (BoW) with TF-IDF transformation

```
param_grid = {
    'model__C': [0.01, 0.1, 1],
    'model__penalty': ['l1', 'l2'],
    'model__solver': ['newton-cg', 'lbfgs', 'liblinear'],
    'vect__ngram_range': [(1, 1), (1, 2), (2, 2)],  # hyperparameter for CountVectorizer
    'tfidf__use_idf': [True, False],  # hyperparameter for TfidfTransformer
    'tfidf__sublinear_tf': [True, False]  # hyperparameter for TfidfTransformer
}
```

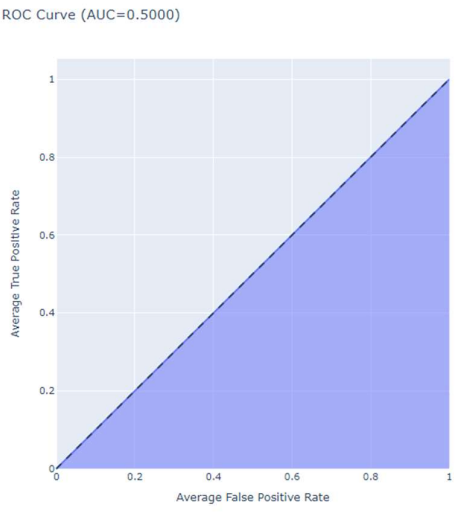
- **Best Parameters** : {'model__C': 0.01, 'model__penalty': 'l1', 'model__solver': 'liblinear', 'tfidf__sublinear_tf': True, 'tfidf__use_idf': True, 'vect__ngram_range': (1, 1)}

5.1.1.4 Model Evaluation

To evaluate the model's best performance, we used the best embedding method and the best parameters by Random Search CV to calculate the performance metrics (accuracy, classification error, sensitivity, precision, specificity and F1_score) and the results are as in the figure below.

- **Base Model**: Logistic Regression
- E**mbedding method**: Bag of Words (BoW) with TF-IDF transformation
- **Best Parameters** : {'model__C': 0.01, 'model__penalty': 'l1', 'model__solver': 'liblinear', 'tfidf__sublinear_tf': True, 'tfidf__use_idf': True, 'vect__ngram_range': (1, 1)}

```
The average accuracy is: 59.35%
The average classification error is: 40.65%
The average sensitivity is: 100.00%
The average precision is: 59.35%
The average specificity is: 0.00%
The average f1 score is: 74.49%
The average AUC Score is: 50.00%
```



After hyperparameter tuning, the sensitivity was 100% but other metrics like accuracy, precision had a steep decrease. 100% sensitivity indicates that the model correctly identifies all positive instances. In

other words, it captures all the cases that belong to a particular class. However, having high sensitivity alone doesn't guarantee a good model, especially if accuracy and precision are low.
Despite high sensitivity, if the model misclassifies a significant number of negative instances (instances not belonging to the positive class), it can lead to low accuracy. The misclassification of negative instances affects the overall correctness of the model.

Hence we will be sticking with the initial model prior to tuning to be the best logistic regression model.

5.1.2 Multinomial Naive Bayes

5.1.2.1 Results for Hypothesis 1

By using Stratified k-Fold Train Test Split, the model iterated through each fold and calculated the following metrics (accuracy, classification error, sensitivity, precision, specificity, F1 score and AUC) for each fold. After appending each metric to a list and computed the average, the results are as follows:

Text Lemmatization Method	Accuracy	Precision	Specificity	Sensitivity	F1	AUC	Classification Error
Nouns	75.44%	72.38%	47.17%	94.80%	82.09%	83.39%	24.56%
Adjectives	75.34%	72.29%	46.93%	94.80%	82.03%	83.27%	24.66%
Verbs	75.47%	72.39%	47.17%	94.85%	82.11%	83.16%	24.53%
Cleaned Text (Base comparison)	75.33	72.28	46.89%	94.80%	82.02%	75.33	24.67%

Based on the results above, the best performing lemmatization is **Verb Lemmatization.**

5.1.2.2 Results for Hypothesis 2

We chose the best performing feature from hypothesis 1 which is the "**Cleaned Text with V lemmatization**" to apply the different embedding techniques on.

Performed hyperparameter tuning for FastText

Base Model Used: Multinomial Naive Bayes

Best Parameters: {'vector_size': 300, 'window': 10, 'min_count': 1, 'sg': 1}

By using Stratified k-Fold Train Test Split, the model iterated through each fold and calculated the following metrics (accuracy, classification error, sensitivity, precision, specificity, F1 score and AUC) for each fold. After appending each metric to a list and computed the average, the results are as follows:

Embedding	**Accuracy**	**Precision**	**Specificity**	**Sensitivity**	**F1**	**AUC**	**Classification Error**
Bag of Words (BoW) with TF-IDF transformation	75.47%	72.39%	47.17%	94.85%	82.11%	83.16%	24.53%
GloVe (version 1) - Vector size 50	72.00%	70.59%	44.94%	90.53%	79.33%	80.58%	28.00%
GloVe (version 1) - Vector size 100	72.92%	72.06%	49.72%	88.81%	79.56%	81.08%	27.08%
GloVe (version 1) - Vector size 200	73.23%	72.95%	52.76%	87.25%	79.46%	81.03%	26.77%
GloVe (version 2)	74.17%	73.41%	53.17%	88.54%	80.27%	82.37%	25.83%
Word2Vec	71.37%	68.50%	35.62%	95.85%	79.89%	82.02%	28.63%
FastText	73.41%	73.22%	53.52%	87.04%	79.53%	79.00%	26.59%
FastText (Hyperparameter Tuned)	71.67%	71.33%	48.71%	87.39%	78.55%	77.58%	28.33%
Doc2Vec	64.22%	63.11%	18.45%	95.56%	76.02%	73.52%	35.78%
Elmo	70.16%	75.27%	64.46%	74.06%	74.66%	77.50%	29.84%

For Multinomial Naive Bayes, even though the recall is the highest for word2vec, the 1% increase in recall compared to Bag of Words (BoW) with TF-IDF transformation is not worth it, as the other metrics decreased by 3-4%. Thus, we picked the second best which is the Bag of Words (BoW) with TF-IDF transformation.

The chosen embedding method is **Bag of Words (BoW) with TF-IDF transformation.**

5.1.2.3 Model Hyperparameter Tuning

We hypertuned the Multinomial Naive Bayes Model with the best performing embedding method to determine the best set of parameters for the Multinomial Naive Bayes Model via Grid Search CV.

- **Base Model:** Multinomial Naive Bayes
- **Embedding method:** Bag of Words (BoW) with TF-IDF transformation

```
# Define the hyperparameter grid
param_grid = {
    'model__alpha': [0.1, 0.5, 1.0, 1.5, 2.0],
    'vect__max_features': [1000, 5000, 10000],  # hyperparameter for CountVectorizer
    'vect__ngram_range': [(1, 1), (1, 2), (2, 2)],  # hyperparameter for CountVectorizer
    'tfidf__use_idf': [True, False],  # hyperparameter for TfidfTransformer
    'tfidf__sublinear_tf': [True, False]  # hyperparameter for TfidfTransformer
}
```

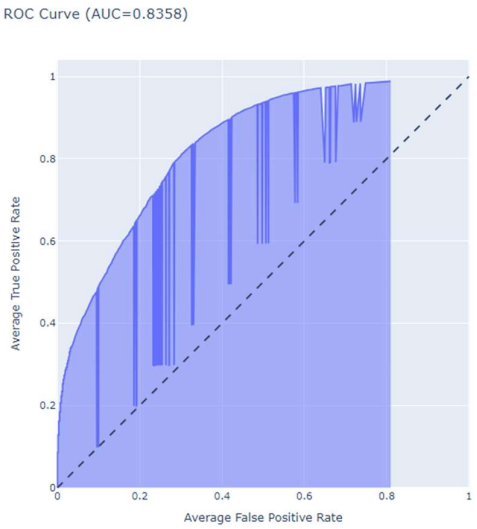
- **Best Parameters:** {'model__alpha': 2.0, 'tfidf__sublinear_tf': True, 'tfidf__use_idf': False, 'vect__max_features': 10000, 'vect__ngram_range': (1, 1)}

5.1.2.4 Model Evaluation

To evaluate the model's best performance, we used the best embedding method and the best parameters by Random Search CV to calculate the performance metrics (accuracy, classification error, sensitivity, precision, specificity and F1_score) and the results are as in the figure below.

- **Base Model:** Multinomial Naive Bayes
- **Embedding method:** Bag of Words (BoW) with TF-IDF transformation
- **Best Parameters :** {'model__alpha': 2.0, 'tfidf__sublinear_tf': True, 'tfidf__use_idf': False, 'vect__max_features': 10000, 'vect__ngram_range': (1, 1)}

```
The average accuracy is: 76.27%
The average classification error is: 23.73%
The average sensitivity is: 94.05%
The average precision is: 73.43%
The average specificity is: 50.30%
The average f1 score is: 82.47%
The average AUC Score is: 83.58%
```



The Sensitivity actually dropped slightly from 94.85 to 94.05. However, it is offset by an increase in accuracy and precision (increase by about 2%)

5.1.3 Random Forest

5.1.3.1 Results for Hypothesis 1

Instead of using Stratified k-Fold Train Test Split, we decided to use the normal Train Test Split for this Random Forest Classifier. The model then iterated through each fold and calculated the following metrics (accuracy, classification error, sensitivity, precision, specificity, F1 score and AUC) for each fold. After appending each metric to a list and computed the average, the results are as follows:

Text Lemmatization Method	Accuracy	Precision	Specificity	Sensitivity	F1	AUC	Classification Error
Nouns	59.35%	59.35%	0.00%	100.00%	74.49%	77.26%	40.65%
Adjectives	59.35%	59.35%	0.00%	100.00%	74.49%	77.64%	40.65%
Verbs	59.35%	59.35%	0.00%	100.00%	74.49%	76.99%	40.65%
Cleaned Text (Base comparison)	59.35%	59.35%	0.00%	100.00%	74.49%	77.26%	40.65%

Based on the results above, the best performing lemmatization is **Cleaned Text.**

5.1.3.2 Results for Hypothesis 2

We chose the best performing feature from hypothesis 1 which is the "**Cleaned Text**" to apply the different embedding techniques on.

Performed hyperparameter tuning for FastText

Base Model Used: Random Forest Classifier

Best Parameters: {'vector_size': 300, 'window': 10, 'min_count': 3, 'sg': 1}

By using Stratified k-Fold Train Test Split, the model iterated through each fold and calculated the following metrics (accuracy, classification error, sensitivity, precision, specificity, F1 score and AUC) for each fold. After appending each metric to a list and computed the average, the results are as follows:

Embedding	Accuracy	Precision	Specificity	Sensitivity	F1	AUC	Classification Error

Bag of Words (BoW) with TF-IDF transformation	59.36%	59.36%	0.03%	100.00%	74.50%	77.70%	40.64%
GloVe (version 1) - Vector size 50	79.75%	80.38%	68.90%	87.18%	83.63%	87.22%	20.25%
GloVe (version 1) - Vector size 200	79.87%	79.44%	66.31%	89.16%	84.02%	87.47%	20.13%
GloVe (version 2)	79.77%	79.96%	67.80%	87.97%	83.77%	87.70%	20.23%
Word2Vec	79.75%	78.79%	64.57%	90.15%	84.09%	87.19%	20.25%
FastText	80.35%	80.29%	68.21%	88.66%	84.26%	88.21%	19.65%
FastText (Hyperparameter Tuned)	80.66%	80.49%	68.50%	88.99%	84.52%	88.33%	19.34%
Doc2Vec	75.18%	75.32%	58.57%	86.56%	80.54%	81.99%	24.82%

For the Random Forest Classifier Model, even though the recall is the highest for BoW with TF-IDF transformation, we believe that we have to weigh both the accuracy and the recall together to come to a more reasonable conclusion. Therefore, the best performing model which takes into account both accuracy and precision would be the FastText model which has been hyperparameter tuned.

The chosen embedding method is **FastText (hyperparameter tuned).**

5.1.3.3 Model Hyperparameter Tuning

We hypertuned the Random Forest Classifier Model with the best performing embedding method to determine the best set of parameters for the Random Forest Classifier via Grid Search CV.

- **Base Model:** Random Forest Classifier
- **Embedding method:** FastText (hyperparameter tuned)

```
# Define the hyperparameter grid
param_grid = {
    'n_estimators': [100, 200, 300, 400],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2']
}
```

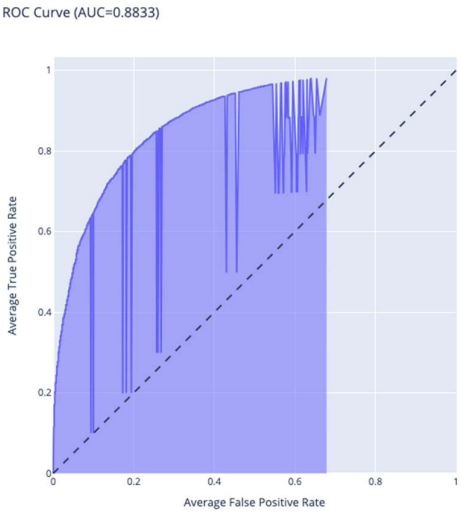
- **Best Parameters:** {'n_estimators':100, 'max_depth': None, 'min_samples_split': 2, 'min_samples_leaf': 4, 'max_features': 'log2'}

5.1.3.4 Model Evaluation

To evaluate the model's best performance, we used the best embedding method and the best parameters by Grid Search CV to calculate the performance metrics (accuracy, classification error, sensitivity, precision, specificity and F1_score) and the results are as in the figure below.

- **Base Model:** Random Forest Classifier
- **Embedding method:** FastText (hyperparameter tuned)
- **Best Parameters :** {'n_estimators':100, 'max_depth': None, 'min_samples_split': 2, 'min_samples_leaf': 4, 'max_features': 'log2'}

```
The average accuracy is: 80.66%
The average classification error is: 19.34%
The average recall is: 88.99%
The average precision is: 80.49%
The average specificity is: 68.50%
The average f1 score is: 84.52%
The average AUC Score is: 88.33%
```



5.1.4 XGBoost

5.1.4.1 Results for Hypothesis 1

By using Stratified k-Fold Train Test Split, the model iterated through each fold and calculated the following metrics (accuracy, classification error, sensitivity, precision, specificity, F1 score and AUC) for each fold. After appending each metric to a list and computed the average, the results are as follows:

Text Lemmatization Method	Accuracy	Precision	Specificity	Sensitivity	F1	AUC	Classification Error

Nouns	80.57%	79.68%	66.35%	90.31%	84.66%	88.46%	80.57%
Adjectives	80.55%	79.63%	66.23%	90.36%	84.65%	88.48%	80.55%
Verbs	80.92%	80.01%	66.96%	90.48%	84.91%	88.60%	80.92%
Cleaned Text (Base comparison)	80.57%	79.68%	66.33%	90.32%	84.66%	88.39%	80.57%

Based on the results above, the best performing lemmatization across all metrics is **Verb Lemmatization.**

5.1.4.2 Results for Hypothesis 2

We chose the best performing feature from hypothesis 1 which is the "**Cleaned Text with V lemmatization**" to apply the different embedding techniques on.

Performed hyperparameter tuning for FastText

Base Model Used: XGBoost

Best Parameters: {'vector_size': 200, 'window': 5, 'min_count': 3, 'sg': 1}

Results of Hyperparameter Tuning of Embedding

After tuning the FastText embedding, accuracy improved 0.29% from 80.55% to 80.84% and Recall improved 1.59% from 84.23% to 85.82%. The tuning took about 35 minutes to run, but the improvements are still pretty small, and hence we will prioritise the actual hyperparameter tuning of the models instead of the embeddings.

By using Stratified k-Fold Train Test Split, the model iterated through each fold and calculated the following metrics (accuracy, classification error, sensitivity, precision, specificity, F1 score and AUC) for each fold. After appending each metric to a list and computed the average, the results are as follows:

Embedding	**Accuracy**	**Precision**	**Specificity**	**Sensitivity**	**F1**	**AUC**	**Classification Error**
Bag of Words (BoW) with TF-IDF transformation	80.92%	80.01%	66.96%	90.48%	84.91%	88.60%	19.08%
GloVe (version 1) - Vector size 50	79.20%	81.32%	71.69%	84.35%	82.80%	86.67%	20.79%

GloVe (version 1) - Vector size 100	79.60%	81.48%	71.79%	84.94%	83.17%	87.47%	20.40%
GloVe (version 1) - Vector size 200	80.18%	81.96%	72.53%	85.42%	83.65%	88.12%	19.82%
GloVe (version 2)	79.73%	81.65%	72.12%	84.95%	83.23%	87.53%	20.27%
Word2Vec	80.56%	82.23%	72.91%	85.81%	83.97%	88.15%	19.44%
FastText	80.55%	82.28%	73.06%	85.68%	83.94%	88.46%	19.45%
FastText (Hyperparameter Tuned)	80.48%	82.13%	72.75%	85.78%	83.91%	88.00%	19.52%
Doc2Vec	75.08%	77.35%	64.91%	82.05%	79.62%	81.80%	24.92%
Elmo	80.15%	81.46%	71.35%	86.18%	83.75%	87.77%	19.85%

For XGBoost, the method that achieved the best results across all metrics is **Bag of Words (BoW) with TF-IDF transformation** ,hence it is our chosen embedding method. We found it surprising that it outperformed the more complicated embedding methods by at least 4%, but it could be due to the current complexity of our data not being as sophisticated.

5.1.4.3 Model Hyperparameter Tuning

We hypertuned the XGBoost model with V-lemmatization and Bag of Words TF-IDF to determine the best set of parameters for the Multinomial Naive Bayes Model via Grid Search CV.

- **Base Model:** XGBoost
- **Embedding method:** Bag of Words (BoW) with TF-IDF transformation

```
# Define the hyperparameter grid
param_grid = {
    'vect__max_df': [0.5, 0.75, 1.0],  # Maximum document frequency for CountVectorizer
    'tfidf__use_idf': [True, False],  # Use TF-IDF or not
    'model__n_estimators': [100, 200, 300],  # Number of boosting rounds
    'model__learning_rate': [0.01, 0.1, 0.2],  # Learning rate
    'model__max_depth': [3, 4, 5],  # Maximum tree depth
    'model__min_child_weight': [1, 2, 3]  # Minimum sum of instance weight needed in a child
}
```

- **Best Parameters:** {'model__learning_rate': 0.01, 'model__max_depth': 4, 'model__min_child_weight': 3, 'model__n_estimators': 100, 'tfidf__use_idf': False, 'vect__max_df': 0.5}

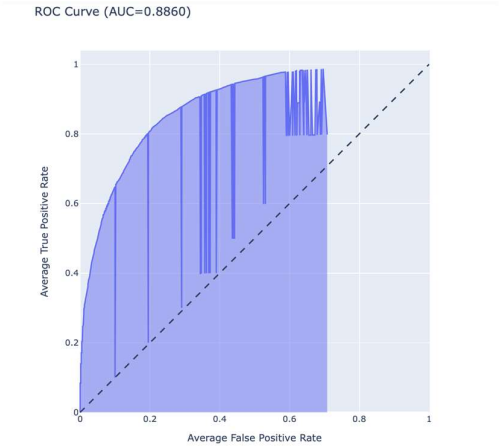
5.1.4.4 Model Evaluation

To evaluate the model's best performance, we used the best embedding method and the best parameters by Random Search CV to calculate the performance metrics (accuracy, classification error, sensitivity, precision, specificity and F1_score) and the results are as in the figure below.

- **Base Model:** XGBoost
- **Embedding method:** Bag of Words (BoW) with TF-IDF transformation

Base Model without Hyperparameter Tuning

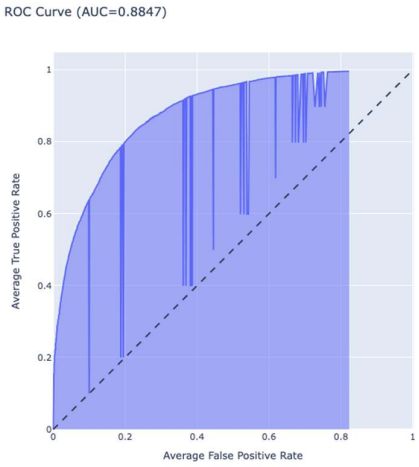
```
The average accuracy is: 80.92%
The average classification error is: 19.08%
The average recall is: 90.48%
The average precision is: 80.01%
The average specificity is: 66.96%
The average f1 score is: 84.91%
The average AUC Score is: 88.60%
```



Model after Hyperparameter Tuning

- **Parameters :** {'model__learning_rate': 0.01, 'model__max_depth': 4, 'model__min_child_weight': 3, 'model__n_estimators': 100, 'tfidf__use_idf': False, 'vect__max_df': 0.5}

```
The average accuracy is: 80.23%
The average classification error is: 19.77%
The average recall is: 91.75%
The average precision is: 78.55%
The average specificity is: 63.40%
The average f1 score is: 84.64%
The average AUC Score is: 88.47%
```



After hyperparameter tuning, we saw that the accuracy fell by 0.69%, but the recall (which we prioritised), improved by 1.27%. We viewed that the increase in recall is worth the sacrifice of 0.69% in accuracy, and decided to use the model after hyperparameter tuning as the best model for XGBoost.

5.1.5 Support Vector Machine (SVM)

5.1.5.1 Results for Hypothesis 1

Instead of using Stratified k-Fold Train Test Split, we decided to use the normal Train Test Split for the Support Vector Machine Model. The model then iterated through each fold and calculated the following metrics (accuracy, classification error, sensitivity, precision, specificity, F1 score and AUC) for each fold. After appending each metric to a list and computed the average, the results are as follows:

Text Lemmatization Method	Accuracy	Precision	Specificity	Sensitivity	F1	AUC	Classification Error
Nouns	59.35%	59.35%	0.00%	100.00%	74.49%	86.44%	40.65%
Adjectives	59.35%	59.35%	0.00%	100.00%	74.49%	86.60%	40.65%
Verbs	59.35%	59.35%	0.00%	100.00%	74.49%	86.47%	40.65%
Cleaned Text (Base comparison)	59.35%	59.35%	0.00%	100.00%	74.49%	86.56%	40.65%

Based on the results above, the best performing lemmatization is **Cleaned Text.**

5.1.5.2 Results for Hypothesis 2

We chose the best performing feature from hypothesis 1 which is the "**Cleaned Text**" to apply the different embedding techniques on.

Performed hyperparameter tuning for FastText

Base Model Used: Support Vector Machine

Best Parameters: {'vector_size': 100, 'window': 15, 'min_count': 3, 'sg': 0}

By using Stratified k-Fold Train Test Split, the model iterated through each fold and calculated the following metrics (accuracy, classification error, sensitivity, precision, specificity, F1 score and AUC) for each fold. After appending each metric to a list and computed the average, the results are as follows:

Embedding	Accuracy	Precision	Specificity	Sensitivity	F1	AUC	Classification Error

Bag of Words (BoW) with TF-IDF transformation	59.35%	59.35%	0.00%	100.00%	74.49%	86.56%	40.65%
GloVe (version 1) - Vector size 50	59.38%	59.37%	0.08%	100.00%	74.51%	81.04%	40.62%
GloVe (version 1) - Vector size 200	59.50%	59.43%	0.36%	100.00%	74.56%	81.99%	40.50%
GloVe (version 2)	59.45%	59.40%	0.24%	100.00%	74.54%	81.60%	40.55%
Word2Vec	59.35%	59.35%	0.00%	100.00%	74.49%	84.42%	40.65%
FastText	59.66%	59.54%	0.77%	100.00%	74.63%	79.96%	40.34%
FastText (Hyperparameter Tuned)	72.15%	69.84%	41.09%	93.43%	79.93%	80.36%	27.85%
Doc2Vec	59.35%	59.35%	0.00%	100.00%	74.49%	76.21%	40.65%

For the Support Vector Machine Model, the best performing model which takes into account both accuracy and precision would be the FastText model which has been hyperparameter tuned.

The chosen embedding method is **FastText (hyperparameter tuned).**

5.1.5.3 Model Hyperparameter Tuning

We hypertuned the Support Vector Machine Model with the best performing embedding method to determine the best set of parameters for the Support Vector Machine Model via Grid Search CV.

- **Base Model:** Support Vector Machine
- **Embedding method:** FastText (hyperparameter tuned)

```
# Define the hyperparameter grid for the SVM model
param_grid = {
    'C': [0.1, 1, 10],  # Regularization parameter
    'gamma': [0.001, 0.01, 0.1],  # Kernel coefficient for 'rbf', 'poly', and 'sigmoid'
    'kernel': ['linear', 'rbf', 'poly', 'sigmoid'],  # Kernel types to test
}
```

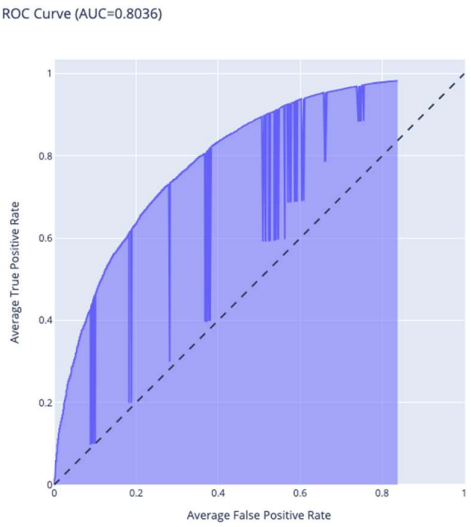
- **Best Parameters:** { 'model_C': 0.1, 'gamma': 0.001, ‘kernel’: ‘rbf’'}

5.1.5.4 Model Evaluation

To evaluate the model's best performance, we used the best embedding method and the best parameters by Grid Search CV to calculate the performance metrics (accuracy, classification error, sensitivity, precision, specificity and F1_score) and the results are as in the figure below.

- **Base Model:** Support Vector Machine
- **Embedding method:** FastText (hyperparameter tuned)
- **Best Parameters :** { 'model_C': 0.1, 'gamma': 0.001, 'kernel': 'rbf'}

```
The average accuracy is: 72.15%
The average classification error is: 27.85%
The average recall is: 93.43%
The average precision is: 69.84%
The average specificity is: 41.09%
The average f1 score is: 79.93%
The average AUC Score is: 80.36%
```



5.2 Results for Deep Learning Neural Network Techniques

5.2.1 Recurrent Neural Network

5.2.1.1 Results for Hypothesis 1

By using normal Train Test Split, the model iterated through each fold and calculated the following metrics (accuracy, classification error, sensitivity, precision, specificity, F1 score and AUC) for each fold. After appending each metric to a list and computed the average, the results are as follows:

Text Lemmatization Method	Accuracy	Precision	Specificity	Sensitivity	F1	AUC	Classification Error
Nouns	72.75%	75.70%	65.90%	77.76%	76.72%	71.83%	27.25%

Adjectives	71.39%	73.21%	60.21%	79.57%	76.26%	69.89%	28.61%
Verbs	71.98%	74.12%	62.27%	79.08%	76.52%	70.68%	28.02%
Cleaned Text (Base comparison)	70.62%	74.18%	64.17%	75.35%	74.76%	69.76%	29.38%

Based on the results above, the best performing lemmatization across all metrics is **Adjectives Lemmatization.**

5.2.1.2 Results for Hypothesis 2

We chose the best performing feature from hypothesis 1 which is the "**Cleaned Text with A lemmatization**" to apply the different embedding techniques on.

Performed hyperparameter tuning for FastText

Base Model Used: Recurrent Neural Network (RNN)

Best Parameters: {'vector_size': 300, 'window': 10, 'min_count': 2, 'sg': 1}

Results of Hyperparameter Tuning of Embedding

After tuning the FastText embedding, accuracy improved by 0.53% from 72.22% to 72.75% and Recall improved by 4.18% from 74.03% to 78.21%. This said, we still go on to explore other types of embeddings and how it can affect the performance of our RNN model.

On top of this, we choose not to use TF-IDF word embeddings in conjunction with Recurrent Neural Networks (RNNs) due to the inherent strengths of RNNs in capturing sequential dependencies and semantic information in data, particularly in natural language sequences. TF-IDF, designed for non-sequential models, lacks the ability to preserve the order of words in a sequence. Introducing TF-IDF features alongside RNN embeddings may increase complexity without a clear benefit, potentially leading to overfitting, especially in datasets with limited samples. Hence, the decision aligns with prioritising the sequential learning capabilities of RNNs and avoiding unnecessary complexity that may not enhance the model's performance on sequential data.

By doing a normal Train Test Split, the model iterated through each fold and calculated the following metrics (accuracy, classification error, sensitivity, precision, specificity, F1 score and AUC) for each fold. After appending each metric to a list and computed the average, the results are as follows:

Embedding	Accuracy	Precision	Specificity	Sensitivity	F1	AUC	Classification Error
GloVe (version 1) - Vector size 50	72.61%	75.95%	64.09%	78.50%	77.21%	77.23%	27.39%
GloVe (version 1) - Vector size 100	72.43%	76.84%	66.72%	76.38%	76.61%	77.61%	27.57%
GloVe (version 1) - Vector size 200	71.95%	76.42%	66.13%	75.97%	76.20%	77.28%	28.05%
GloVe (version 2)	70.69%	73.09%	57.53%	79.80%	76.30%	74.86%	29.31%
Word2Vec	72.47%	76.38%	65.45%	77.33%	76.85%	77.48%	27.53%
FastText	72.22%	77.88%	69.62%	74.03%	75.91%	71.82%	27.78%
FastText (Hyperparameter Tuned)	72.75%	76.28%	64.85%	78.21%	77.23%	71.53%	27.25%
Doc2Vec	74.24%	76.73%	64.51%	80.98%	78.80%	76.53%	25.76%
Elmo	73.09%	76.29%	64.51%	79.03%	77.64%	77.93%	26.91%

For RNN, the method that achieved the best results across all metrics is **Doc2Vec** ,hence it is our chosen embedding method. This shows that Doc2Vec, being a contextual embedding method that considers the context of the words within a document, is particularly beneficial for our classification task, especially when it involves understanding of the overall context in which the words appear to identify if the sentence is classified as stressed or not stressed. On the other hand, embeddings like GloVe, Word2Vec, and FastText are static and do not capture the context in which words appear.

5.2.1.3 Model Hyperparameter Tuning

We hypertuned the RNN model with A-lemmatization and Doc2Vec to determine the best set of parameters for the Multinomial Naive Bayes Model via Grid Search CV.

- **Base Model:** Recurrent Neural Network (RNN)
- **Embedding method:** Doc2Vec

```
# Define the hyperparameter search space
param_dist = {
    'embedding_dim': [50, 100, 150],
    'num_units': [32, 64, 128],
    'learning_rate': [0.01, 0.001, 0.0001]
}
```

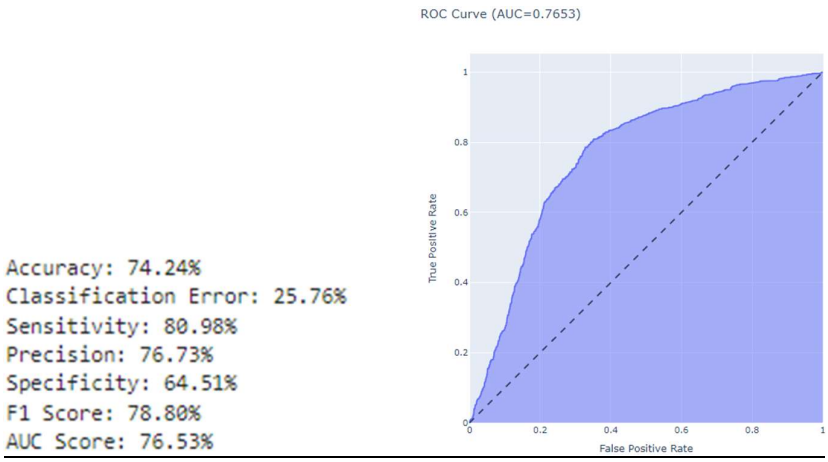
- **Best Parameters:** {'num_units': 64, 'learning_rate': 0.0001, 'embedding_dim': 100}

5.2.1.4 Model Evaluation

To evaluate the model's best performance, we used the best embedding method and the best parameters by Random Search CV to calculate the performance metrics (accuracy, classification error, sensitivity, precision, specificity and F1_score) and the results are as in the figure below.

- **Base Model:** Recurrent Neural Network (RNN)
- **Embedding method:** Doc2Vec

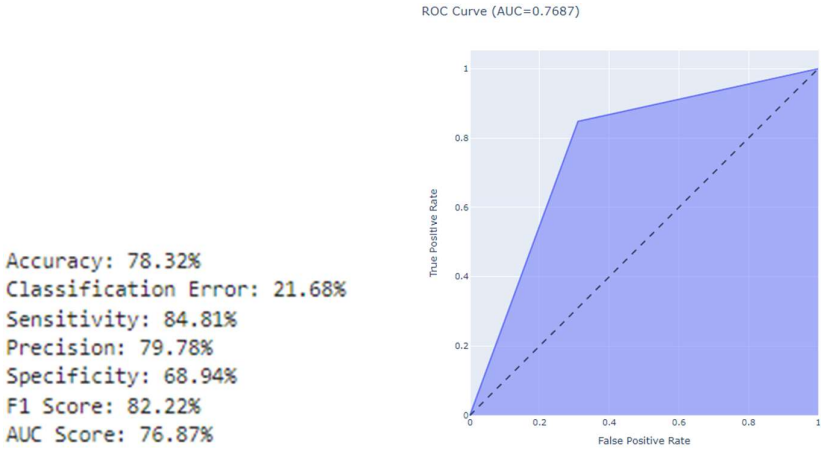
Base Model without Hyperparameter Tuning




```
Accuracy: 74.24%
Classification Error: 25.76%
Sensitivity: 80.98%
Precision: 76.73%
Specificity: 64.51%
F1 Score: 78.80%
AUC Score: 76.53%
```

Model after Hyperparameter Tuning

- **Parameters :** {'num_units': 64, 'learning_rate': 0.0001, 'embedding_dim': 100}



After hyperparameter tuning, we saw that the accuracy increased by 4.08%, and the recall (which we prioritised), improved by 3.83%. F1 Score also improved by 3.42%, from 78.80% to 82.22%. Since accuracy, recall, and F1 score improved, we decided to use the model after hyperparameter tuning as the best model for RNN.

5.2.2 Long Short-Term Memory (LSTM)

5.2.2.1 Results for Hypothesis 1

As LSTM did not accept raw text as inputs, we decided to transform it into Term Frequency-Inverse Document Frequency (TF-IDF) format. We then did a 80-20 train-test split fit into our model to obtain its performance metrics (accuracy, classification error, sensitivity, precision, specificity, F1 score and AUC). The results are as follows:

Text Lemmatization Method	**Accuracy**	**Precision**	**Specificity**	**Sensitivity**	**F1**	**AUC**	**Classification Error**
Nouns	79.67%	81.34%	73.64%	84.09%	82.69%	78.86%	20.33%
Adjectives	79.67%	81.30%	73.56%	84.15%	82.70%	78.85%	20.33%
Verbs	79.74%	81.33%	73.56%	84.27%	82.77%	78.91%	20.26%
Cleaned Text (Base comparison)	79.67%	81.41%	73.81%	83.97%	82.67%	78.89%	20.33%

Based on the results above, the best performing lemmatization is **Verb Lemmatization.**

5.2.2.2 Results for Hypothesis 2

We chose the best performing feature from hypothesis 1 which is the "**Cleaned Text With V lemmatization**" to apply the different embedding techniques on.

Performed hyperparameter tuning for FastText

Base Model Used: LSTM

Best Parameters: {'vector_size': 300, 'window': 10, 'min_count': 1, 'sg': 1}

By using Stratified k-Fold Train Test Split, the model iterated through each fold and calculated the following metrics (accuracy, classification error, sensitivity, precision, specificity, F1 score and AUC) for each fold. After appending each metric to a list and computed the average, the results are as follows:

Embedding	Accuracy	Precision	Specificity	Sensitivity	F1	AUC	Classification Error
Bag of Words (BoW) with TF-IDF transformation	81.00%	83.28%	75.05%	85.08%	84.16%	88.46%	19.00%
GloVe (version 1) - Vector size 50	79.06%	80.14%	68.78%	86.10%	82.99%	85.97%	20.94%
GloVe (version 1) - Vector size 100	80.26%	81.31%	70.88%	86.69%	83.91%	87.55%	19.74%
GloVe (version 1) - Vector size 200	81.23%	82.25%	72.48%	87.23%	84.65%	88.61%	18.77%
GloVe (version 2)	80.23%	81.23%	70.71%	86.74%	83.89%	87.54%	19.77%
Word2Vec	80.83%	81.81%	71.66%	87.12%	84.36%	88.19%	19.17%
FastText	80.63%	81.48%	70.80%	87.37%	84.25%	88.44%	19.37%
FastText (Hyperparameter Tuned)	80.81%	81.47%	70.72%	87.72%	84.42%	88.43%	19.20%
Doc2Vec	73.03%	74.58%	58.64%	82.88%	78.48%	79.34%	26.97%
Elmo	81.88%	83.72%	75.46%	86.28%	84.96%	89.69%	18.12%

For LSTM, we decided to go with **FastText(Hyperparameter Tuned)**. This is because it had the highest sensitivity (recall) while still having good results for the other performance metrics. Close

competitors were Word2Vec and Glove (version 1) -Vector size 200, coming in 2nd and 3rd. But since FastText achieved similar results for accuracy and/or F1 score compared to them, we decided to stick with FastText(Hyperparameter Tuned).

5.2.2.3 Model Hyperparameter Tuning

We hypertuned the LSTM model with V-Lemmatization and FastText(Hyperparameter Tuned) to determine the best set of parameters.

- **Base Model:** LSTM
- **Embedding method:** FastText (Hyperparameter Tuned)

```
units_list = [50, 100, 150]
activation_lstm_list = ['sigmoid', 'tanh', 'relu']
activation_dense_list = ['sigmoid', 'tanh', 'relu']
optimizer_list = [tf.keras.optimizers.Adam, tf.keras.optimizers.RMSprop, tf.keras.optimizers.Adagrad]
learning_rate_list = [0.001, 0.01, 0.1]
epochs_list = [5, 10]
batch_size_list = [32, 64]
```

- **Best Parameters:** {'units': 100, 'activation_LSTM': 'relu', 'activation_Dense': 'sigmoid', 'optimizer':'Adam', 'learning_rate': 0.01, 'epochs': 10, 'batch_size': 64}

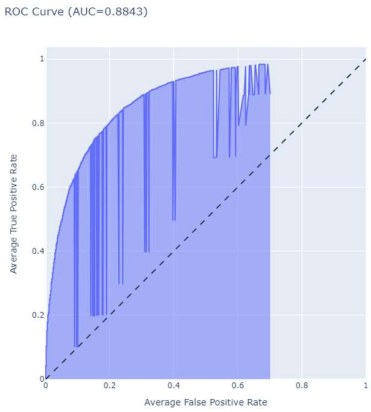
5.2.2.4 Model Evaluation

To evaluate the model's best performance, we used the best embedding method and the best parameters to calculate the performance metrics (accuracy, classification error, sensitivity, precision, specificity and F1_score) and the results are as in the figure below.

- **Base Model:** LSTM
- **Embedding method:** FastText (Hyperparameter Tuned)

Base Model without Hyperparameter Tuning

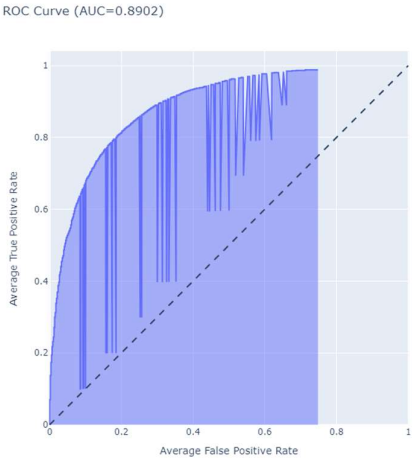
The average accuracy is: 80.81%
The average classification error is: 19.20%
The average sensitivity is: 87.72%
The average precision is: 81.47%
The average specificity is: 70.72%
The average f1 score is: 84.42%
The average AUC Score is: 88.43%



Model after Hyperparameter Tuning

- **Parameters:** {'units': 100, 'activation_LSTM': 'relu', 'activation_Dense': 'sigmoid', 'optimizer':'Adam', 'learning_rate': 0.01, 'epochs': 10, 'batch_size': 64}

The average accuracy is: 81.08%
The average classification error is: 18.92%
The average sensitivity is: 88.31%
The average precision is: 81.56%
The average specificity is: 70.53%
The average f1 score is: 84.69%
The average AUC Score is: 89.02%



After hyperparameter tuning, we saw an increase in performance for accuracy from 80.81% to 81.08%, sensitivity (recall) from 87.72% to 88.31%, and F1 score from 84.42% to 84.69%. We can see that although the hyperparameter tuning improved our model, the increase in performance was not very significant.

5.3 Summary and Limitations

Hyperparameter Tuning

Some of the models that we used were very computationally intensive for this NLP problem. The team attempted to hyperparameter tune using GridSearchCV and RandomSearchCV, while computationally feasible. Due to this, our team decided to limit the number of iterations done for our hypertuning process due to the time and computational constraints. This may have resulted in suboptimal hypertuned models

and might not reflect their true potential. We attributed this to a lack of domain knowledge and experience in determining which parameters are key, since we primarily turned to online research papers online for guidance.

Recall as Key Metric

Furthermore, for this project, we decided to focus on recall to determine the best model as false negatives, which are people that are stressed but undiagnosed, are a much higher cost as compared to misdiagnosed people who do not have stress. As such we hyper parameter tuned the models based on recall score. While we felt that the focus was justified, it did lead to certain models having precision scores that were too low as the tuning was heavily skewed towards recall. This can be clearly seen in our Linear Support Vector Machine Model (0.93 recall but 0.69 precision). Due to time constraints of the project, we were unable to change the hyperparameter tuning to be based on either F1 score or AUC score which finds a balance between precision and recall. Hence, in the future, we will look into improving the hyper parameter tuning process by choosing a balanced metric.

Generalisation of Findings

Another key consideration is that our dataset is primarily from Twitter(X) and Reddit. Whilst it has given us meaningful insights in stress detection of social media, it does bear certain limitations. The demographics of the users of these social media platforms might belong to specific demographics or showcase unique user behaviours that might not be true for other users. For example, if the users of Reddit tend to be 30 and below, it might not be able to tell us much about stress for users above the age of 30. The way the platforms are set up, like the anonymity or limits such as character limits of each tweet, might also affect what the user posts. A discussion the team had was that different cultures and geographical locations might have different stress factors and also different usage of platforms. An observation amongst our peers as students in Singapore was the increase in career related stress. In countries that face such a situation, maybe it would also be relevant to include data from sites like Linkedin.

6. Conclusion and Future Work

6.1 Model Evaluation - Hypothesis 3 : Which model performs best for the classification of text?

	Precision	Recall	F1-Score
Logistic Regression	0.82	0.91	0.86
Multinomial NB	0.73	0.94	0.83
XGBoost	0.80	0.92	0.85
Linear SVM	0.69	0.93	0.80
Random Forest	0.80	0.89	0.85
LSTM	0.81	0.88	0.84
RNN	0.80	0.85	0.82

These are the final results we have achieved with the best parameters of each model. Judging solely based on Recall, Multinomial Naive Bayes obtained the best score of 0.94. However, as mentioned earlier on in this report, we wanted to optimise and get the best recall whilst still maintaining a reasonable precision. The team decided that a precision of 0.73 was too low, and was willing to take a drop of 0.02 in recall for the 0.07 increase in precision. Hence, our final model was **XGBoost using V-Lemmatization and BOW with TF-IDF transformations**.

We found it intriguing that the classical machine learning models did better than the deep learning machine models, considering that Natural Language Processing is quite a complicated task. This highlighted the importance of understanding our dataset and utilising the methods that are most appropriate for this particular set of characteristics for the dataset. This process requires a continuous exploration of the data as well as the process of finding the golden mean between computational complexity and model performance.

It does seem that in this case with the current complexity of our dataset, the principle of Occam's Razor applies where the simplest solution is the best.

6.2 Improvements from Literature Review

6.2.1 Literature review 1

From literature review 1, we learnt that the small data size could have prevented a higher accuracy for their models, and that their data was not diverse in the sense that it only came from a single source of social media platform.

To address this, we leveraged a larger dataset from Kaggle and Github, incorporating diverse data from Reddit and Twitter (see section 3.1). Comparing identical models with the same embedding methods across both projects revealed significant improvements across all metrics (notably, a slight drop in precision and accuracy for SVM was offset by a substantial increase in recall). This proves that a larger dataset and diverse data could increase the performance of machine learning models.

Embedding Method	Project	Classifier	F1 score	Recall	Precision	Accuracy
BoW*TF-IDF	Literature Review 1	Logistic Regression	0.70	0.60	0.64	0.65
		XGBoost	0.60	0.66	0.63	0.66
		SVM	0.69	0.64	0.60	0.64
	Our Project	Logistic Regression	0.86	0.91	0.81	0.82
		XGBoost	0.85	0.90	0.80	0.81
		SVM	0.74	1.00	0.59	0.59

6.2.2 Literature review 2

From Literature review 2, we learnt that the lack of embedding methods may have caused the models to not be as effective as they could have been.

Hence, we decided to explore different types of embedding methods and apply them into our machine learning models. Taking Multinomial Naive Bayes as an example, a comparative analysis across both projects revealed that the BoW*TF-IDF embedding method stood out, exhibiting notable improvements in F1 score and Recall—given our emphasis on recall. Through our exploration of different embedding methods for each machine learning model, we observed distinct results. In most instances, models with applied embedding methods consistently outperformed those without, underscoring the positive impact of embedding methods on model performance.

Classifier	Project	Embedding Method	F1 score	Recall	Precision	Accuracy
Multinomial Naive Bayes	Literature Review 2	None	0.78	0.78	0.79	0.79
	Our Project	None	0.82	0.95	0.72	0.75
		BoW* TF-IDF (Best) (After Hypertuning)	0.82	0.95	0.73	0.76

6.2.3 Literature review 3

From Literature review 3, we learnt that word embeddings can help capture emotional nuances and complexities, and that cross-validation techniques like k-fold can help us understand our models better especially in terms of overfitting and generalisation.

Similar to Literature review 2, we can see that the use of word embeddings do indeed help our models achieve better performance. We have also used k-fold cross validation for our hypothesis testing and overall evaluation of our hypertuned model to accurately gauge the performance of our models.

6.3 Future Work

Looking ahead, there is still much work that can be done in terms of further improving the performance and scope of our classification models. Our team has identified some potential avenues for future work which we look forward to exploring:

1. Transformer Architecture Model

   The Transformer Architecture Model is a deep learning model based on self-attention mechanisms, which allows the model to selectively weigh the importance of different parts of the input data during data processing. This allows the model to learn long-range dependencies without the need for recurrent connections, allowing the training of the model to be more efficient.

2. Emoticon-based Sentiment Analysis

   Emoticons, or Emojis, are often used as a simple and effective way to convey one's emotions online. In this project, we actually removed all symbols and non-alphabetical characters due to the complexity of analysing each unique emoticon(some might not have much support in the dataset). Using emoticon-based sentiment analysis to determine the emotional expression of a

comment allows the model to consider its underlying context and nuance. The model can then accurately determine the intended sentiment of the comment and produce more accurate results and possibly interesting insights.

3. Oversampling through Data Augmentation

   In this project, our team chose to address the problem of an imbalanced dataset by undersampling the majority class. Another method we can use is to instead oversample the minority class (i.e. toxic comments) by artificially creating modified copies using the existing training dataset. This will allow the model to learn more about the underlying data of toxic comments and can improve its accuracy in classifying whether a comment is toxic or not. However, it is also important to note that oversampling can lead to overfitting of the model. Hence, techniques like regularisation and early stoppage should be used in conjunction with oversampling to avoid a model that has been overfitted.

4. Integration of individual models using ensemble methods.

   The use of ensemble methods can bring about a number of advantages. Namely, improved accuracy, reduced overfitting and robustness to noise and outliers. However, one thing to potentially take into account is how computationally intensive this might be. A simple and less computationally intensive method would be a hard voting ensemble, where we could combine different models together to make a prediction.

5. Leveraging Domain-Specific Lexicons (Jurek et al., 2015)

   Domain-specific lexicons are essentially specialised terminology for a particular area of interest or topic. Such lexicons can be curated to capture the unique linguistic nuances and expressions prevalent in online platforms such as Reddit and Twitter. By incorporating lexicons that encompass informal language, slang, and domain-specific terminology related to stress and emotional well-being, the models can better contextualise and interpret user comments. This nuanced understanding enables us to identify subtle emotional cues that may contribute to stress-related content. Moreover, leveraging domain-specific lexicons allows for a more informed analysis of sentiment and emotional tone, further refining our models' ability to discern stress-related language patterns in the diverse and dynamic landscape of social media.

References

Amazon Web Services. (2023). How XGBoost Works. Amazon Sagemaker. https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost-HowItWorks.html

Chaware, S. M., Makashir, C., Athavale, C., Athavale, M., & Baraskar, T. (2020). Stress detection methodology based on Social Media Network: A proposed design. *International Journal of Innovative Technology and Exploring Engineering*, *9*(3), 3489–3492. https://doi.org/10.35940/ijitee.b7537.019320

Dutta, Mimi. (2023, November 9). Word2Vec For Word Embeddings -A Beginner's Guide. https://www.analyticsvidhya.com/blog/2021/07/word2vec-for-word-embeddings-a-beginners-guide/

Factory, C. (n.d.). Singapore social media statistics and facts 2023. Commission Factory Affiliate Marketing Blog. https://blog.commissionfactory.com/affiliate-marketing/singapore-social-media-statistics

Foundation, E. N. (n.d.). Social Media Vent. LinkedIn. https://www.linkedin.com/pulse/social-media-vent-endnowfoundation/

Inamdar, S., Chapekar, R., Gite, S., & Pradhan, B. (2023). Machine learning driven mental stress detection on reddit posts using natural language processing. Human-Centric Intelligent Systems, 3(2), 80–91. https://doi.org/10.1007/s44230-023-00020-8

Joshi, P. (2022, June 23). A Step-by-Step NLP Guide to Learn ELMo for Extracting Features from Text. https://www.analyticsvidhya.com/blog/2019/03/learn-to-use-elmo-to-extract-features-from-text/

Jurek, A., Mulvenna, M. D., & Bi, Y. (2015). Improved lexicon-based sentiment analysis for social media analytics. Security Informatics (Berlin), 4(1), 1-. https://doi.org/10.1186/s13388-015-0024-x

Khanna, C. (2021, February 10). Text pre-processing: Stop words removal using different libraries. Medium. https://towardsdatascience.com/text-pre-processing-stop-words-removal-using-different-libraries-f20bac19929a

Kalra, authorAditi S., Sofiah, 18 September 2023Arina, Sofiah, 13 September 2023Arina, Sunil, 13 September 2023Priya, & Chan, 13 September 2023Tracy. (2023, March 7). Stressed in Singapore: Rising cost of living is triggering wellbeing issues. Human Resources Online. https://www.humanresourcesonline.net/stressed-in-singapore-rising-cost-of-living-is-triggering-wellbeing-issues

Khare, P. (April 2). Deep Learning for NLP: Word2Vec, Doc2Vec, and Top2Vec Demystified. https://medium.com/mlearning-ai/deep-learning-for-nlp-word2vec-doc2vec-and-top2vec-demystified-3842b4fad5c9

Kharwal, A. (2021, July 7). *Classification Report in Machine Learning*. Data Science | Machine Learning | Python | C++ | Coding | Programming | JavaScript. https://thecleverprogrammer.com/2021/07/07/classification-report-in-machine-learning/

Khushaktov, M. F. (2023, August 26). *Introduction random forest classification by example*. Medium. https://medium.com/@mrmaster907/introduction-random-forest-classification-by-example-6983d95c7b91

Krithika, V. (2023, January 19). Introduction to FastText Embeddings and its Implication. https://www.analyticsvidhya.com/blog/2023/01/introduction-to-fasttext-embeddings-and-its-implication/

Pennington, J., Socher, R., Manning, D.C., (2014). GloVe: Global Vectors for Word Representation https://nlp.stanford.edu/projects/glove/

Piduguralla, S. (2023, September 8). Understanding the Sigmoid Function in Logistic Regression: Mapping Inputs to Probabilities. LinkedIn. https://www.linkedin.com/pulse/understanding-sigmoid-function-logistic-regression-pidugurallа/

Rajani, K. (2023a, March 3). Human stress prediction. Kaggle. https://www.kaggle.com/datasets/kreeshrajani/human-stress-prediction

Srinidhi, S. (2023, March 15). Lemmatization in Natural Language Processing (NLP) and machine learning. Built In. https://builtin.com/machine-learning/lemmatization

Thisishusseinali. (2023, June 23). Stresssense: Stress Detection System 🧠. Kaggle. https://www.kaggle.com/code/thisishusseinali/stresssense-stress-detection-system/notebook

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

Wikimedia Foundation. (2023, November 4). *Support Vector Machine*. Wikipedia. https://en.wikipedia.org/wiki/Support_vector_machine