University of the Philippines Manila
College of Arts and Sciences
Department of Physical Sciences and Mathematics

Using NLP and Binary Classification for Predicting Suicidal Tendencies Based on Textual Input

A special problem in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Computer Science

Submitted by:

Ronell Mathew R. Cruz
June 2023

Permission is given for the following people to have access to this SP:

Available to the general public	Yes
Available only after consultation with author/SP adviser	No
Available only to those bound by confidentiality agreement	No

ACCEPTANCE SHEET

The Special Problem entitled "Using NLP and Binary Classification for Predicting Suicidal Tendencies Based on Textual Input" prepared and submitted by Ronell Mathew R. Cruz in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

Vincent Peter C. Magboo, M.D., M.Sc.
Adviser

EXAMINERS:

	Approved	**Disapproved**
1. Avegail D. Carpio, M.Sc.		
2. Richard Bryann L.Chua, Ph.D. (*cand.*)		
3. Perlita E. Gasmen, M.Sc.(*cand.*)		
4. Ma. Sheila A. Magboo, Ph.D.(*cand.*)		
5. Marbert John C. Marasigan, M.Sc.(*cand.*)		
6. Geoffrey Solano, Ph.D.		

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

Vio Jianu C. Mojica , M.Sc.
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences
and Mathematics

Marie Josephine M. De Luna, Ph.D.
Chair
Department of Physical Sciences
and Mathematics

Maria Constancia O. Carrillo, Ph.D
Dean
College of Arts and Sciences

Abstract

Suicide is death caused by injuring oneself with the intent to die. Suicidal tendency is a certain set of behavior that an individual exhibits when being suicidal, which includes posting cryptic messages in social media. This is a study that uses the Suicide and Depression Dataset and applies NLP and preprocessing techniques to transform the data. Machine learning models are then used to predict the class of the entries. Logistic Regression is the best performing model with an accuracy score of 93.24%. The best performing model is integrated and used in a Discord bot application.

Keywords: Machine Learning, Natural Language Processing, Hashing Vectorizer, Tokenizer, Stemming

Contents

List of Figures

List of Tables

I. Introduction

A. Background of the Study

Suicide is death by injuring oneself with the intent to die, while a suicide attempt is when someone harms themselves with any intent to end their life, but they do not die as a result of their actions. Suicide is a leading cause of death and is a serious public health issue.

According to WHO, more than 700,000 people due to suicide every year, which converts to 1 in every 100 deaths. Crucially, it is also tagged as the fourth leading cause of death in people aged 15-29 years old. In the Philippines, there were 44 cases of suicide per month during the period from January to October 2022, and of the 40 suicide deaths reported, 23 (which is about 58%) were below 30 years old, according to data from Atlantic Fellows. This stems mostly from the lack of access to mental health care for young Filipinos, especially in remote areas, which denies early detection and intervention. Attempted suicide and suicide death rates has seen a sharp increase during the COVID-19 pandemic.

The link between suicide and mental disorders (depression, and alcohol use disorders) is well-established in high income countries. Many suicides however, occur impulsively in moments of crisis with a breakdown in the ability to deal with life stresses, such as a financial crisis, breakup, loss of a family member, chronic illness, or academic and workspace stress. Experiencing conflict, disaster, violence, abuse, or loss and a sense of isolation are also strongly associated with suicidal behaviour, Suicide rates are also high in people who experience discrimination (LGBTQ+ members, migrants, prisoners, etc.). However as of currently, the strongest risk factor in suicide is a previous suicide attempt.

Since the introduction of social media people have been increasingly using online means to express their suicidal tendencies. Teenagers in particular, use social channels to express suicidal intentions, seek advice from online forums, and even participate in suicide pacts. Anonymity is provided via online communication, allowing people to openly communicate their thoughts and experiences as they currently undergo through it in the real world [1].

These online platforms enable early suicide detection and prevention. Analyzing and examining online posts for suicidal tendencies prompted the development of new forms of prospective health care solutions and early suicide detection systems. This is accomplished by detection of suicidal ideas in text entries using NLP (Natural Language Processing) methodology and machine learning techniques. NLP is an interdisciplinary domain which is concerned with understanding natural languages as well as using them to enable human-computer interaction. It combines linguistic and artificial intelligence to understand human inputs on a natural linguistic level. Machine learning on the other hand, is a major branch of artificial intelligence that deals with the design and development of algorithms capable of identifying complex patterns of experimental data without considering a predetermined equation as a model and making intelligent decisions.

The study uses NLP along with multiple binary classification techniques (a subset of machine learning) in order to build a model that can take user inputted text as an input and predict suicidal intentions or tendencies. The study uses a labelled data set containing posts that are binary classified as suicide, or non-suicide. Furthermore, the study includes the generated best-performing model into Discord. Discord is an American VoIP and instant messaging social platform. It is chosen as the platform for the study because it is currently an emerging social media platform that is now commonly used as a voice and messaging platform, specifically for groups of people. According to Statista, there are 456 million active users on the platform as of 2022. It is also a platform that is primarily used by younger audiences, with data from Similar Web showing 35.49% of total Discord users being with the 18-24 years old age group and 32.23% from 25-34 years old age group. Discord is also chosen as it offers a rich, open source API that is commonly used among the platform by users, and that has the necessary features to implement the application. The study utilizes Discord bot from the developer portal in order to implement a classifier that can detect suicidal intent from messages across a certain server.

B. Statement of the Problem

Suicide and suicide attempts cause serious emotional, physical, and economic impacts. People who attempt suicide and survive may experience serious injuries that can have long-term effects on their health. They may also experience depression and other mental health concerns. Also, suicide can be detrimental to the well-being and health of related people such as family, friends, relatives, colleagues, and the general community itself, which can lead to them fostering thoughts and ideas related to suicide as well.

One of the key interventions for preventing suicide is early detection. Early identification of suicidal thoughts, will be crucial in managing and following up on people undergoing a mental disorder, or a drastic life event that they are having a difficulty handling by themselves. Fostering socio-emotional life skills especially in adolescents, along with accessibility to services which caters to one's mental health is also important factors of reducing risk of suicide. However, this is not the case for most people. Especially in countries like the Philippines, according to Atlantic Fellows, lack of access and awareness on mental health problems is an increasing risk factor for suicidal thoughts especially among people aged 15-29. Social media however, provides a form of therapy and a space for people to express their feelings, and experiences, and sometimes cryptic messages that may be a cry for help, or suicidal expression. This platform allows us to extend our reach further to people experiencing suicidal thoughts, and allows us to create solutions that can provide early intervention for a lot of people.

As such, the project aims to utilize the online platform in detecting suicidal tendencies in user messages on a social messaging platform. The project aims to use NLP for processing the text, and to fit the processed text into a binary classification model. The project then aims to embed the generated model in a

Discord bot as a suicide detection tool in an actual social messaging platform.

C. Objectives of the Study

The objectives of this study is divided into two parts:

1. Create a machine learning model:
   (a) Apply preprocessing steps to the data to be processed using NLP. Steps include lowercasing of all text, removal of emoticons and of all non-alphabet characters, and label encoding.
   (b) Apply NLP methodology to the preprocessed data. Removing of stop words, tokenization of strings, and using hashing vectorizer to convert the data numerically to be inputted in to the binary classifier models.
   (c) Use 80-20 train-test split for the data, and train the model using Logistic Regression, Naive Bayes, and XGBoost machine learning techniques for binary classification.
   (d) Test the models using the data and derive accuracy, precision, and recall.
   (e) Identify the best performing model based on the aforementioned metrics.
   (f) Export the best performing model to be used for the Discord detection bot
2. Use the Discord Developer features for model integration:
   (a) Create a Discord bot using the Discord Developer Portal and grant necessary permissions, access, integrations, and auth tokens.
   (b) Once the bot is created, import the generated model via the Discord API, and integrate within the message function to evaluate every message sent.

(c) Detect and flag users that sent a message expressing suicidal intent.

(d) Configure the Discord bot using API commands to send messages of flagged users to a text channel that can only be accessed by certain roles for notification (e.g moderator)

D. Significance of the Project

This study serves as an application of NLP and machine learning techniques in order to aid in suicidal intent detection, and an application to a common social messaging platform (Discord) for possible early action and intervention:

1. **Discord users** – One of the main benefactors of this project. Users that are currently undergoing through a phase of breakdown has a chance of using this social messaging platform in order to express suicidal and depressive thoughts. The messages can then be detected by the Discord bot and the user will receive necessary help. Discord moderators or the roles assigned to oversee the notifications, will then be able to identify users within their server that might be in need of support and assistance, and will prompt them to create an evidence-based decision to provide intervention for the user.

2. **Psychologists and Psychiatrists** – They are considered as the experts in the field. This project can serve as an aiding resource for early detection of suicidal intent, and therefore enabling the experts to provide appropriate intervention measures as early as possible.

3. **Students of Medicine and Computing Sciences** – This project can be viewed as an application of computing sciences in the health sciences field, which will be inevitably more prominent within the upcoming years, and therefore will serve as an example or a point of improvement that can be studied further.

4. **Instructors of Medicine and Computing Sciences** – This project can serve as a reference material to demonstrate potential solutions that address medical problems and also the capabilities of the computing sciences to provide grounds and a platform to apply the solutions.

5. **Researchers** – This study serves as an alternative proposed solution that can inspire researchers to take on a similar approach with problems with the same nature. Also, this can be a foundation that can motivate further study and research within the area, and also motivate researchers to further enhance, modify, or add new features to this currently existing project.

E. Scope and Limitations

1. This project can only process text that are written in the English language, accommodating other languages for the model would require further study and a new approach.

2. This end product of this project is only usable on the social platform Discord, which is chosen as it provides a rich, open source API library that suits the requirements of the application, and its compatibility with the Python language, which is the same language used to develop the machine learning models.

F. Assumptions

1. The users that use the system mainly converse using the English language.

2. The server owner, which the application grants access to control the channel where the application sends the messages, uses the system with proper discretion and consideration of the users.

II. Review of Related Literature

A. Social Media and Suicide

The introduction of social media within the recent years have provided a new avenue for people to express their feelings, thoughts, hardships, and experiences. Along with this, it has also been a place for people to post their suicidal thoughts and tendencies. The provided anonymity of social media is something that people find as a safe space and it encourages them to be more open about the life events they are going through [1]. Among all of these, social media users also find a sense of sympathy and relation with users that harbor similar thoughts or ideas, a study conducted by Swedo et al. [2] found that exposure to these groups that are labelled as ”suicide clusters” significantly increase suicidal ideation and attempt rates. Such examples of these are pacts that are formed within these clusters that convinces the affected users more to push through with their suicidal tendencies. A study by Miyagi et al. [3] exhibits an example of this setting, the study analyzed suicidal posts as a true sign or call of help for affected people, and can be an avenue of early intervention. But at the same time, can be abused by people to further convince the affected users to push through with the ideations, which happened within the Zama suicide pact slayings. This is precisely one of the ordeals that the study is trying to solve. This study shows that the efficacy of social media intervention can be undermined by these kind of events and is motivating researchers and experts to work towards finding methods of earlier detection of suicidal tendencies in social media.

B. Suicidal Text Detection using NLP and Machine Learning

Given the context of suicidal posts in social media, there has been a focus of studying the possibility of utilizing social media posts for early detection and intervention of suicidal thoughts. Ji et al. [4] specifically reviewed several machine learning approaches in detecting suicidal ideation through text. It compared four classification methods namely logistic regression, random forest, gradient boosting decision tree, and XGBoost. In addition, it also used a deep learning approach to the problem. The study found that all methods produced satisfactory results, but also pointed out the main hurdles of this certain area such as data deficiency, lack of intention understanding, annotation bias, and data imbalance. Chaterjee et al. [5] conducted a similar study, putting focus on the preprocessing of text and emphasizing the importance of extracting only the important parts necessary for the classification algorithm. It used similar classification methods as the aforementioned study, with the exception of using support vector machine in place of gradient boosting decision tree. The study expounded on the prevalence of the use of social media in suicidal thoughts expression, and the importance of reinforcement through the use of modern technology and tools to empower early detection. Wang et al. [6] another study that employed NLP and machine learning for suicide thoughts detection, did a comparison study involving logistic regression, support vector machine, and convolutional neural network (CNN) in predicting suicidal ideation. While CNN gained the most favorable results in the study, the study was quite limited on the feature extraction and text preprocessing part. The classification methods that are employed for most of the literature were support vector machine, logistic regression, random forest, and boosting methods. As for the study by Nordin et al. [7], they emphasized on the possible increase in performance that is brought by using ensemble learning techniques, in this case, gradient boosting. Their study has shown that gradient boosting has outperformed random forest in terms of model performance, but only by a very small margin (0.02%). This shows that classification methods perform mostly similar to ensemble learning techniques. The works mentioned support the foundation of using NLP with machine learning for suicide text detection. In particular, the use of classification methods and deep learning were very prevalent among the studies. The preprocessing steps and the application of NLP are emphasized to be the most cruicial steps of the process, and that the models perform similar in terms of performance and is very dependent on data quality, data imbalance, as well as feature selecting. Among the studies, the common limitations are the lack of data that can further introduce a more complex way of interpreting suicidal thoughts, the sheer data imbalance because of the magnitude of social media, possibilities of annotation bias, and also the fact that NLP itself is still relatively limited in terms of understanding robust human expressions.

C. Application of Suicidal Text Detection in Social Media Platforms

The ultimate goal of developing a suicidal detection tool is to integrate it with social media or messaging platforms to enable early intervention for possibly affected users. Wu et al. [8], used NLP techniques such as Long Short-Term Memory and Bidirectional Encoder to develop a model for suicide detection. This was then applied to common social media sites in Taiwan. The study however, also lists out the limitations of the said integration. Initially, NLP only processes textual language, so its detection is only limited to posts that it can process, meaning that suicidal intent that is posted through means of images, and video graphics is out of range for detection. Lastly, machine learning still has a limited understanding of complex human language, and therefore longer and more detailed textual posts become very hard to interpret using their developed model. A study by Sarsam et al. [9] offers some solution to the complexity limitation. Their model utilized NRC Affect Intensity Lexicon which classifies the text into eight major emotions, and then the results show that suicidal ones are exclusive to fear, sadness, and negative sentiments. However, this approach would require text to be lexicon-accurate, and excessive filtering of words may lead to an over-generalization of text to be related to suicide even when it does not fit the criteria. There is also a more domain-specific application and approach to this idea, as presented in the study done by Morrow et al. [10]. The study used the NLP based classifier to extract vocabulary and text closely associated to suicidal risk in clinical notes from electronic health records (EHR). This however, while being more accurate, is limited to being too controlled. The text data that would be extracted from this would be far too rigid, since there is a set convention on how to record text from a medical specific domain, and therefore the approach would see less success when applied to a social media environment where the textual inputs are dynamic and free form. There are many applications of the suicide detection tool idea as a concept to allow for early detection and intervention. However, legal issues in data privacy means that these models cannot be simply integrated to social media platforms and start reading users' textual inputs. Discord however, offers a feature that allows these plugins to be integrated easily, with the permissions, to be assessed and granted by the users themselves, and precisely the reason why the platform is currently viewed by the current study to be the most suitable for the objectives.

III. Theoretical Framework

A. Suicide

Suicide is death caused by injuring oneself with the intent to die. Other associated terms with suicide are suicide attempt, and suicide intent. Suicide attempt is when a person make an effort to end one's life but is unable to do so, while suicide intent is when a person expresses signs and risk factors of committing suicide.

These are the common suicide risk factors

1. **Individual Risk Factors**
    (a) Previous suicide attempt
    (b) History of depression and other mental illnesses
    (c) Serious illness such as chronic pain
    (d) Criminal/legal problems
    (e) Job/financial problems or loss
    (f) Impulsive or aggressive tendencies
    (g) Substance use
    (h) Current or prior history of adverse childhood experiences
    (i) Sense of hopelessness
    (j) Violence victimization and/or perpetration
2. **Relationship Risk Factors**
    (a) Bullying
    (b) Family/loved one's history of suicide
    (c) Loss of relationships
    (d) High conflict or violent relationships
    (e) Social isolation

3. **Community Risk Factors**

    (a) Lack of access to healthcare

    (b) Suicide cluster in the community

    (c) Stress of acculturation

    (d) Community violence

    (e) Historical trauma

    (f) Discrimination

4. **Societal Risk Factors**

    (a) Stigma associated with help-seeking and mental illness

    (b) Easy access to lethal means of suicide among people at risk

    (c) Unsafe media portrayals of suicide

Along with that, here are the most common preventive measures for suicide.

1. **Individual Protective Factors**

    (a) Effective coping and problem-solving skills

    (b) Reasons for living (for example, family, friends, pets, etc.)

    (c) Strong sense of cultural identity

2. **Relationship Protective Factors**

    (a) Support from partners, friends, and family

    (b) Feeling connected to others

3. **Community Protective Factors**

    (a) Feeling connected to school, community, and other social institutions

    (b) Availability of consistent and high quality physical and behavioral healthcare

4. **Societal Protective Factors**

   (a) Reduced access to lethal means of suicide among people at risk

   (b) Cultural, religious, or moral objections to suicide

B. Natural Language Processing

Natural language processing (NLP) refers to the branch of computer science—and more specifically, the branch of artificial intelligence or AI—concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.

In this study, we incorporate various techniques of NLP in processing our text data

1. **Tokenization** - Tokenization is essentially splitting a phrase, sentence, paragraph, or an entire text document into smaller units, such as individual words or terms called tokens. This allows us to perform feature extraction and selection on the dataset and to quantify the tokens for the machine learning models.

2. **Stemming** - Stemming is a natural language processing technique that lowers inflection in words to their root forms, hence aiding in the preprocessing of text, words, and documents for text normalization. The study will utilize Porter stemming for processing the tokenized data. This algorithm is considered as a truncating algorithm, which is more focused on removing affixes from text. Figure 4 provides an example on how the stemmer processes text.

3. **Text Normalization** - A technique that reduces a given text to its canonical form. For this study, several steps to normalize text are applied.

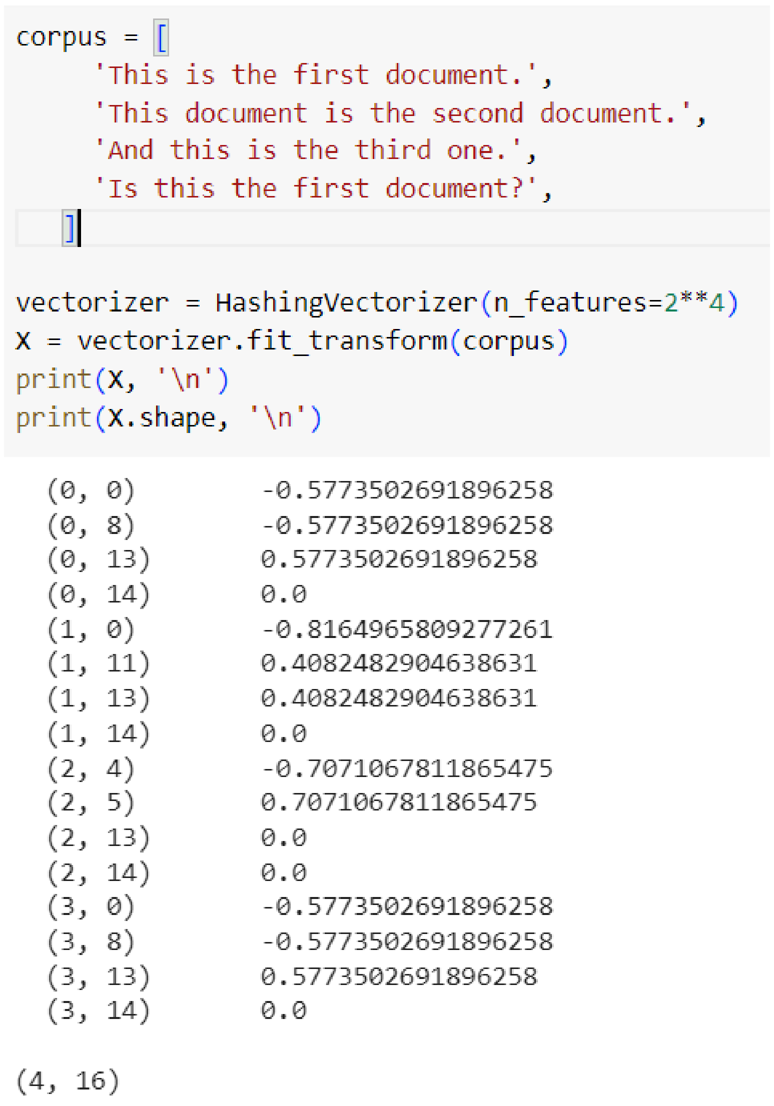
   (a) Removal of Emoticons and other Non-Alphabetical Characters

   (b) Porter stemming algorithm for finding the root word

   (c) Lowercasing of all text for uniformity

4. **Removal of Stop Words** - Stop words are words in any language which are most commonly used, but does not add contextual significance to the meaning of the text. Removing stop words from the text would allow for a more efficient processing of data, and allows the models to concentrate on the more important words that predict our target variable.

5. **Hashing Vectorizer** - Hashing Vectorizer converts a text document into a matrix of token occurrences, which is a sparse matrix which contains frequency of the tokens (the words). It is also known as the hashing trick as it utilizes feature hashing to store token occurences.



```
corpus = [
     'This is the first document.',
     'This document is the second document.',
     'And this is the third one.',
     'Is this the first document?',
   ]

vectorizer = HashingVectorizer(n_features=2**4)
X = vectorizer.fit_transform(corpus)
print(X, '\n')
print(X.shape, '\n')
```

```
  (0, 0)	-0.5773502691896258
  (0, 8)	-0.5773502691896258
  (0, 13)	0.5773502691896258
  (0, 14)	0.0
  (1, 0)	-0.8164965809277261
  (1, 11)	0.4082482904638631
  (1, 13)	0.4082482904638631
  (1, 14)	0.0
  (2, 4)	-0.7071067811865475
  (2, 5)	0.7071067811865475
  (2, 13)	0.0
  (2, 14)	0.0
  (3, 0)	-0.5773502691896258
  (3, 8)	-0.5773502691896258
  (3, 13)	0.5773502691896258
  (3, 14)	0.0

(4, 16)
```

Figure 1: Sample Code for Hashing Vectorizer (From sklearn's website)

C. Machine Learning Models

1. **Logistic Regression**- Logistic regression is a supervised learning algorithm used to predict a dependent categorical target variable. It is an algorithm which predicts the output of a categorically dependent variable. Therefore, Logistic Regression works well with binary classification problems, where the outcome is dichotomous. The classifier also gives out the probabilistic values that lie between 0 and 1, which is the binary values that represents the two categories of the target variable. Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.

2. **Naive Bayes** - The Naïve Bayes classifier is a popular supervised machine learning algorithm used for classification tasks such as text classification. It belongs to the family of generative learning algorithms, which means that it models the distribution of inputs for a given class or category.

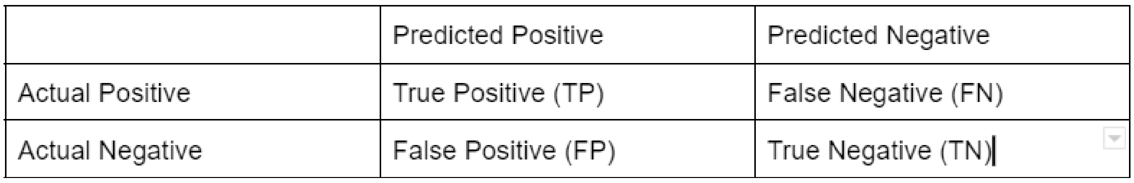
   In statistics, naive Bayes classifiers are considered as simple probabilistic classifiers that apply Bayes' theorem. This theorem is based on the probability of a hypothesis, given the data and some prior knowledge. The naive Bayes classifier assumes that all features in the input data are independent of each other, which is often not true in real-world scenarios. However, despite this simplifying assumption, the naive Bayes classifier is widely used because of its efficiency and good performance in many real-world applications.

3. **Extreme Gradient Boosting** - Extreme Gradient Boosting, or also known as XGBoost is an optimized distributed gradient boosting library designed for efficient and scalable training of machine learning models. It is an ensemble learning method that combines the predictions of multiple weak models to produce a stronger prediction. XGBoost stands for "Extreme Gradient

Boosting" and it has become one of the most popular and widely used machine learning algorithms due to its ability to handle large datasets and its ability to achieve state-of-the-art performance in many machine learning tasks such as classification and regression.

D. Performance Metrics and Indicators

The performance metric is a quantified assessment that determines the success of a certain process. For the context of Machine Learning, the models are evaluated using the metrics accuracy, precision, f1 score, and recall. These values would mainly be computed using information derived from the confusion matrix



	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Figure 2: The Confusion Matrix

1. Accuracy - A metric that is commonly used in classification problems and to evaluate a classification model's performance. It is an evaluation metric that measures the correct predictions made by a model relative to the total predictions that it made.

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

Figure 3: Accuracy Formula

2. Precision - A common metric that is used to evaluate a classification model's performance. It is the ratio of the correctly identified positives to the total number of positives (whether identified correctly or not). It gives us a clearer idea if the model's positive predictions are actually positive.

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

Figure 4: Precision Formula

3. Recall - A metric that evaluates how much of the actual positives have been correctly identified by the model. It provides us the information about how well the model is at determining the positive values of the data.

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

Figure 5: Recall Formula

4. F1 Score - Is a machine learning evaluation metric that similarly with accuracy, evaluates the general performance of a model. The difference is, F1 score utilizes the precision and recall scores to generate a better quantification of the model's performance.

$$F1\ Score = \frac{2(Precision)(Recall)}{Precision + Recall}$$

Figure 6: F1 Score Formula

E. Discord Developer Portal

Discord is a free communications app that lets you share voice, video, and text chat with friends, game communities, and developers. It is a VoIP methodology-based platform which delivers voice and multimedia sessions over internet protocol networks. The users can interact within their own specific groups known as ”servers”. The servers have their own moderators, which can set roles and manage users, as well as create multiple text and voice channels. Discord also allows the users to contribute to the application via creating bots with rich functionalities, or even entertainment from games through a platform called the Discord Developer Portal

The Discord Developer Portal is an environment with rich functionalities to create entities that can be utilized in various ways within the Discord platform. It features Bots and Apps creation, which are empowered by API and OAuth2 integrations, and Game Creation using Rich Presence and GameSDK. Discord was chosen over other social media platforms because the API features are the most suitable for the application of the study. It offers a comprehensive Python library which allows for easy modularization of commands, and automation of responses via API. It is also an open-source tool that can be easily accessed by developers, as well as a tool that is commonly used within the platform itself, allowing for a more practical solution to the problem. The library being Python also helps provide security for versions and compatibility for the models when imported, as the models were also trained using the Python language.

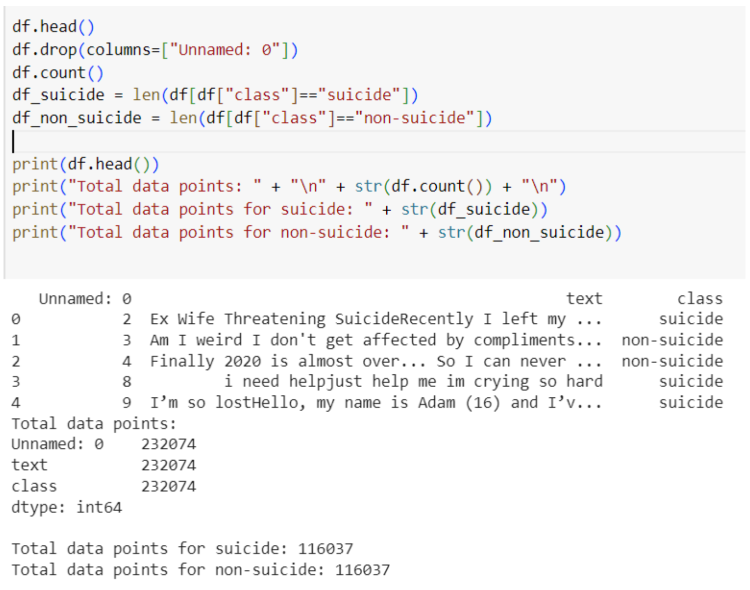
For the study, the developers uses the Bot Python API feature to develop the suicide detection bot. The API feature allows for integration within a specific Discord server and to execute necessary commands for suicide detection.

IV. Design and Implementation

A. Description of the Dataset

The study uses a labelled dataset that was collected from two subreddits of the Reddit platform named "Depression" (January 1, 2009 - January 2, 2021) and "SuicideWatch" (December 16, 2008 - January 2, 2021). The non-suicide entries are gathered from the "Teenagers" subreddit.

The dataset consists of 232,074 data points, which then have 4 columns, a blank-headed column, but contains the index, an column named "unnamed: 0", a column named "text" which contains the actual text from the post, and lastly a column named "class" which contains whether the given text is labelled as suicide or non-suicide. The data is also equally split between data labelled suicide and non suicide with both having 116,037 entries.



```
df.head()
df.drop(columns=["Unnamed: 0"])
df.count()
df_suicide = len(df[df["class"]=="suicide"])
df_non_suicide = len(df[df["class"]=="non-suicide"])

print(df.head())
print("Total data points: " + "\n" + str(df.count()) + "\n")
print("Total data points for suicide: " + str(df_suicide))
print("Total data points for non-suicide: " + str(df_non_suicide))
```

```
   Unnamed: 0                                               text        class
0           2  Ex Wife Threatening SuicideRecently I left my ...      suicide
1           3  Am I weird I don't get affected by compliments...  non-suicide
2           4  Finally 2020 is almost over... So I can never ...  non-suicide
3           8          i need helpjust help me im crying so hard      suicide
4           9  I’m so lostHello, my name is Adam (16) and I’v...      suicide
Total data points:
Unnamed: 0    232074
text          232074
class         232074
dtype: int64

Total data points for suicide: 116037
Total data points for non-suicide: 116037
```

Figure 7: Dataset Information

B. Data Visualization

1. Histogram

The x-axis within this histogram is the amount of words per data point, which corresponds to the length of the entire text when counted by words.

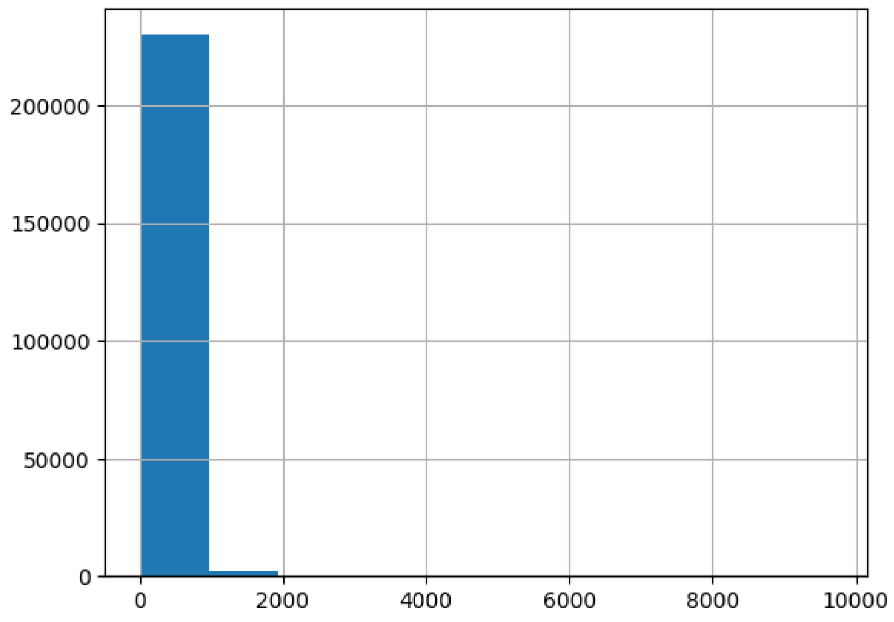


Figure 8: Histogram of Word Length for Entire Dataset

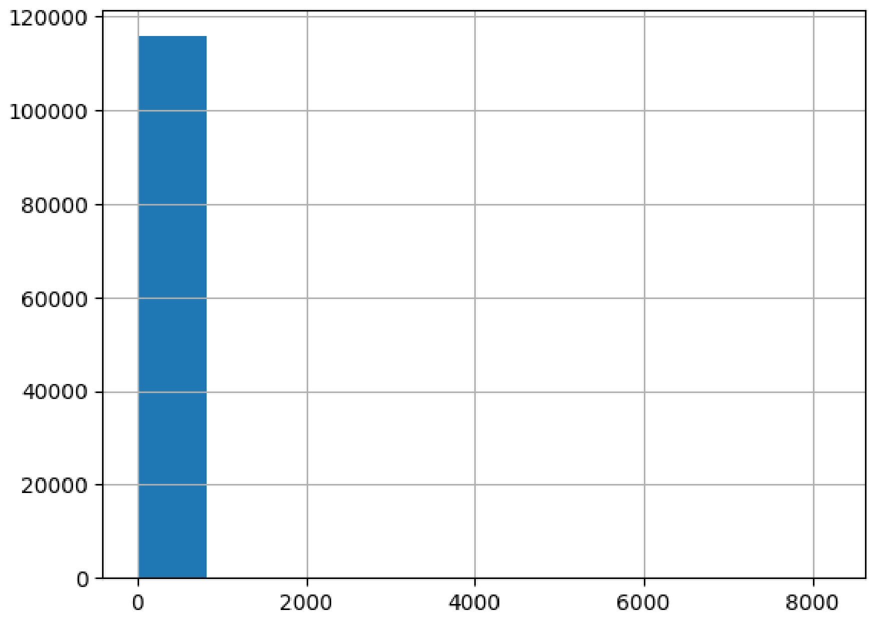


Figure 9: Histogram of Word Length for Entire Non-Suicide Entries

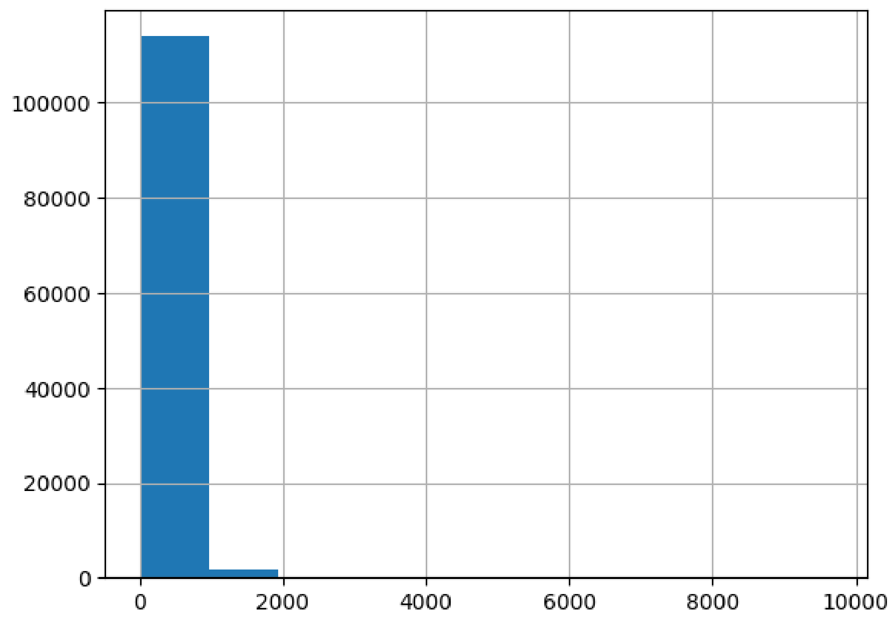


Figure 10: Histogram of Word Length for Entire Suicide Entries

For all of the categories, the text majorly lies below 1000 words. Suicide entries have more entries that lies between 1000 to 2000 words compared to non-suicide entries but only by an insignificant amount. It is also seen by the generated range of the histogram that there are some text entries which exceed 2000 words, but are miniscule in amount compared to the majority of the data points.

2. Wordcloud

   Wordcloud is a visual representation of word frequency and word prominence from a source text. In the study, Wordcloud visualization is used to have a general idea of the commonly occuring text within the overall text, suicide text, and non-suicide text.

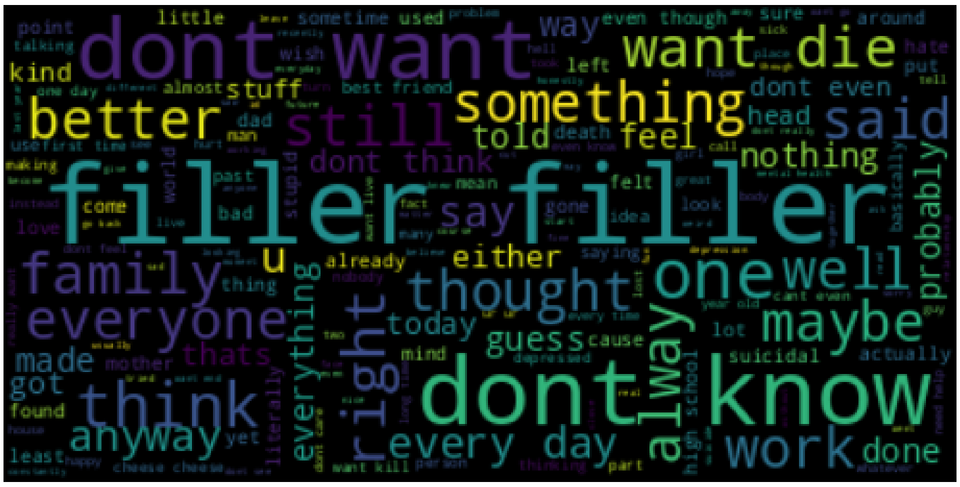


Figure 11: Wordcloud for Entire Dataset

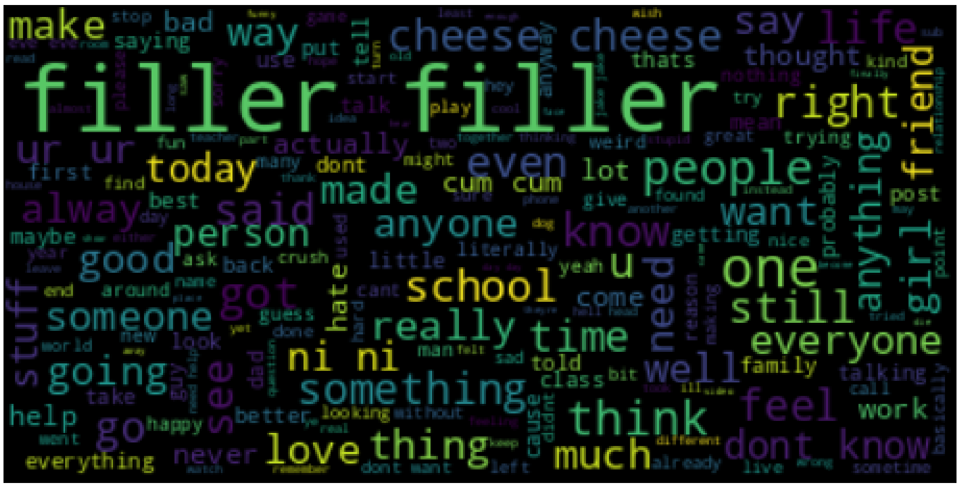


Figure 12: Wordcloud for Non-Suicide Entries

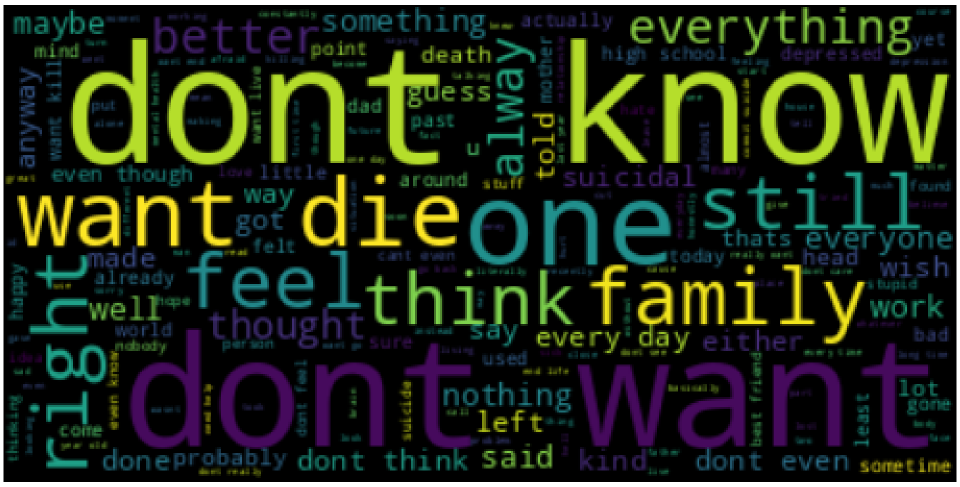


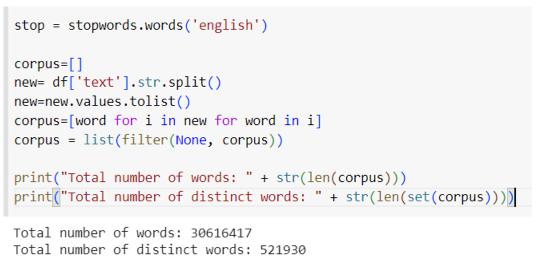
Figure 13: Wordcloud for Suicide Entries

This wordcloud shows the prominent words that appears all throughout the entire dataset, and also the word prominence for the two classes the model is targeting to classify. There are words that appear on all the wordclouds but differ in frequency. Words such as "want" appear in all the wordclouds but appear most frequent in the suicide dataset. Words such as "love", "think", and "don't" are also some of the words that appear on all the wordclouds with varying frequency.

There are also words that are significantly prominent or uniquely prominent to the suicide wordcloud that does not appear in the non-suicide wordcloud and vice versa. Words from the suicide wordcloud such as "die", "nothing" and words from the non-suicide wordcloud such as "filler", "school", "friend" are some of the words that appear in greater prominence from their respective counterpart wordcloud. This shows that word frequency is a significant to determine suicide or non-suicide entries.

C. Corpus Construction

The next step is creating a list of all the words within the text to be analyzed. By simply splitting the text and not applying any other preprocessing techniques, we get a total of 30,616,417 words and 521,930 distinct words within the dataset.

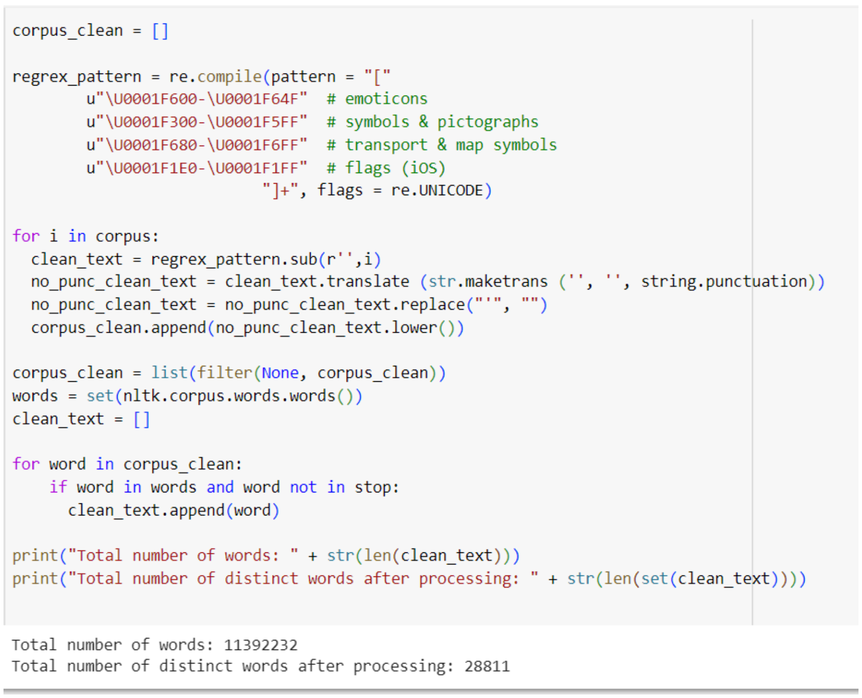


```
stop = stopwords.words('english')

corpus=[]
new= df['text'].str.split()
new=new.values.tolist()
corpus=[word for i in new for word in i]
corpus = list(filter(None, corpus))

print("Total number of words: " + str(len(corpus)))
print("Total number of distinct words: " + str(len(set(corpus))))
```

```
Total number of words: 30616417
Total number of distinct words: 521930
```

Figure 14: Corpus Construction (Without Preprocessing)

Several preprocessing techniques are then applied to the data. The emoticons within the text are removed using regular expressions. The text is then trimmed of all punctuation marks attached to them and then all lower cased. Finally, the corpus list is filtered out for empty entries, non-english words, and stop words. After processing, we reduce the total number of text in the list to 11,392,232 words and 28,811 distinct words.



```
corpus_clean = []

regrex_pattern = re.compile(pattern = "["
        u"\U0001F600-\U0001F64F"  # emoticons
        u"\U0001F300-\U0001F5FF"  # symbols & pictographs
        u"\U0001F680-\U0001F6FF"  # transport & map symbols
        u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                          "]+", flags = re.UNICODE)

for i in corpus:
  clean_text = regrex_pattern.sub(r'',i)
  no_punc_clean_text = clean_text.translate (str.maketrans ('', '', string.punctuation))
  no_punc_clean_text = no_punc_clean_text.replace("'", "")
  corpus_clean.append(no_punc_clean_text.lower())

corpus_clean = list(filter(None, corpus_clean))
words = set(nltk.corpus.words.words())
clean_text = []

for word in corpus_clean:
    if word in words and word not in stop:
      clean_text.append(word)

print("Total number of words: " + str(len(clean_text)))
print("Total number of distinct words after processing: " + str(len(set(clean_text))))
```

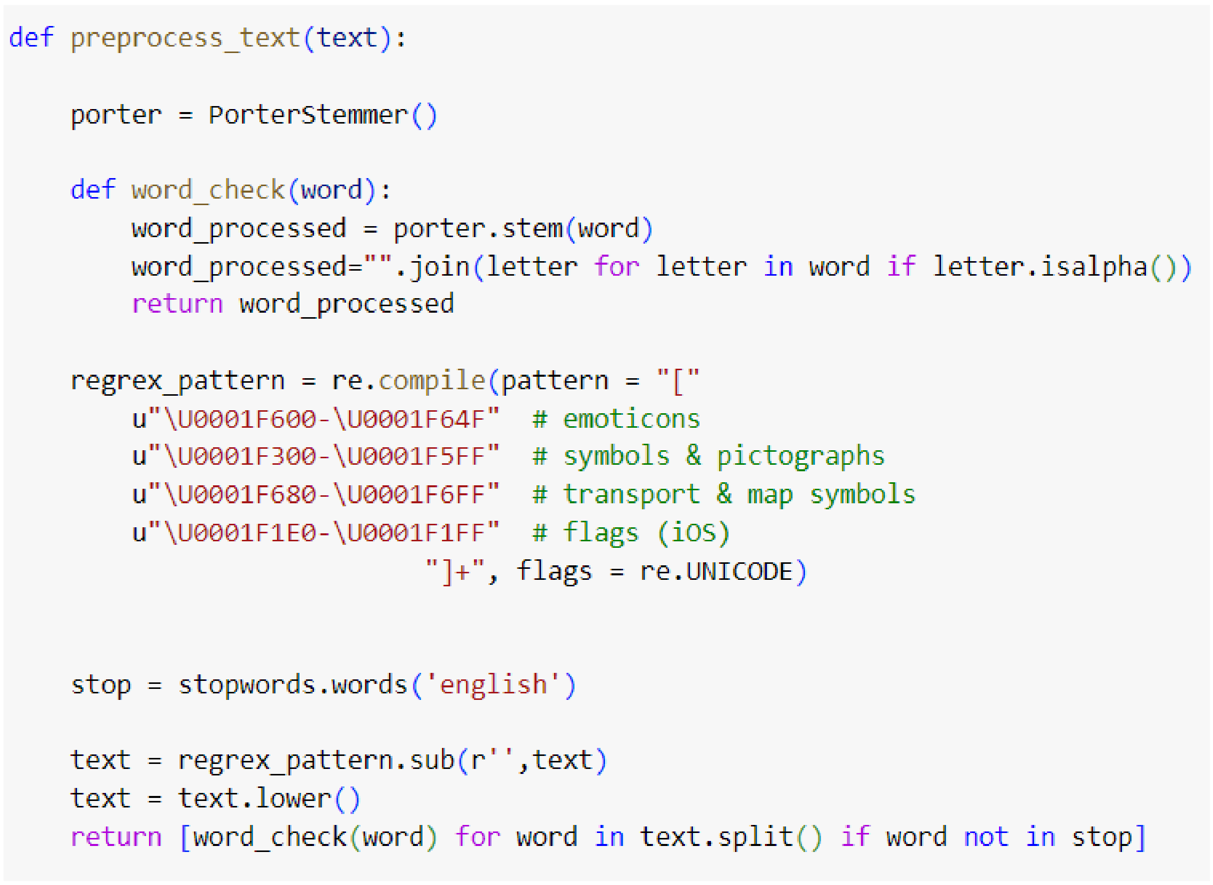
```
Total number of words: 11392232
Total number of distinct words after processing: 28811
```

Figure 15: Corpus Construction (After Processing)

D. Preprocessing Techniques

1. Setting Up the Tokenizer Function

The tokenizer function splits the text and apply all necessary filtering and processing. The tokenizer function would first remove all emoticons from the text, and lowercase everything. The next step is to apply Porter stemming, which is a process of obtaining the root word for the text, in order to derive the proper frequency from words with similar ideas. The last steps would be to remove punctuation and non-alphabet characters and filter out stop words that would not be weighted for the study.

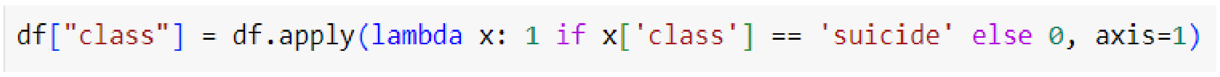


```python
def preprocess_text(text):

    porter = PorterStemmer()

    def word_check(word):
        word_processed = porter.stem(word)
        word_processed="".join(letter for letter in word if letter.isalpha())
        return word_processed

    regrex_pattern = re.compile(pattern = "["
        u"\U0001F600-\U0001F64F"  # emoticons
        u"\U0001F300-\U0001F5FF"  # symbols & pictographs
        u"\U0001F680-\U0001F6FF"  # transport & map symbols
        u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           "]+", flags = re.UNICODE)


    stop = stopwords.words('english')

    text = regrex_pattern.sub(r'',text)
    text = text.lower()
    return [word_check(word) for word in text.split() if word not in stop]
```

Figure 16: The Tokenizer Function

2. Label Encoding

   Label encoding is simply assigning a numerical value for the categories that the model is predicting. For the study, it is simply assigning binary values 0 for non-suicide and 1 for suicide

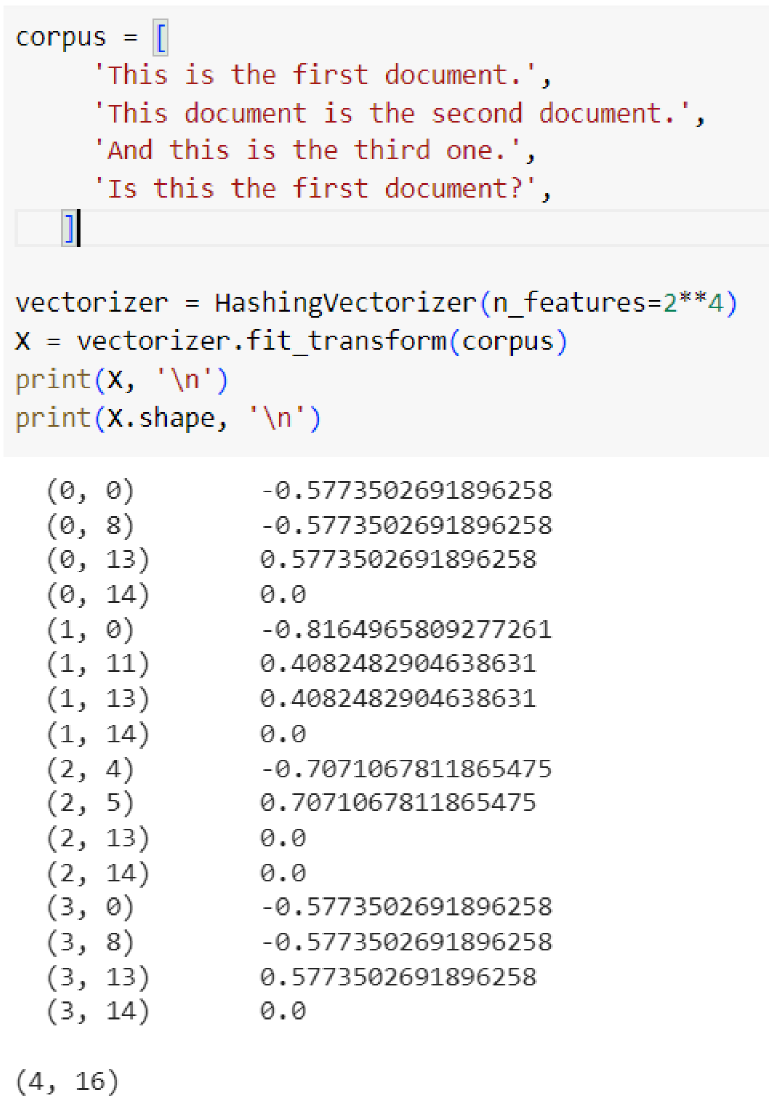


```python
df["class"] = df.apply(lambda x: 1 if x['class'] == 'suicide' else 0, axis=1)
```

Figure 17: Label Encoding

3. Feature Extraction

For the feature extraction, the study used sklearn's Hashing Vectorizer. Hashing Vectorizer converts a text document into a matrix of token occurrences, which is a sparse matrix which contains frequency of the tokens (the words). It is also known as the hashing trick as it utilizes feature hashing to store token occurences.



```
corpus = [
     'This is the first document.',
     'This document is the second document.',
     'And this is the third one.',
     'Is this the first document?',
   ]

vectorizer = HashingVectorizer(n_features=2**4)
X = vectorizer.fit_transform(corpus)
print(X, '\n')
print(X.shape, '\n')
```

```
  (0, 0)        -0.5773502691896258
  (0, 8)        -0.5773502691896258
  (0, 13)       0.5773502691896258
  (0, 14)       0.0
  (1, 0)        -0.8164965809277261
  (1, 11)       0.4082482904638631
  (1, 13)       0.4082482904638631
  (1, 14)       0.0
  (2, 4)        -0.7071067811865475
  (2, 5)        0.7071067811865475
  (2, 13)       0.0
  (2, 14)       0.0
  (3, 0)        -0.5773502691896258
  (3, 8)        -0.5773502691896258
  (3, 13)       0.5773502691896258
  (3, 14)       0.0

(4, 16)
```

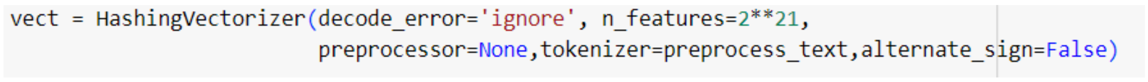
Figure 18: Sample Code for Hashing Vectorizer (From sklearn's website)

As seen in the sample code, the Hashing Vectorizer converted the sample corpus text to a sparse matrix consisting of 4 documents and 16 features

(4 rows and 16 columns). Hashing Vectorizer's main advantage is it is low memory scalable, making it ideal for working with relatively large datasets, such as the one used in the study, with 11,393,232 tokens to store. The hashing also works as a frequency counter, and would be a great tool for quantifying the data.

The main problem when using Hashing Vectorizer (or any other hashing algorithms in general) is the possibility of hash collisions. The issue is addressed in two ways within the study. First, the study utilized a large feature space in order to make sure that each token will have its own hash function. Second, the parameter "alternate_sign" is set to False in order to avoid having negative values due to hash collisions which would eventually affect the performance of the machine learning models.

This step utilizes our tokenizer function as the text processor. The hashing vectorizer would then be setup with 2**21 n_features (which would be large enough to ensure no hash collisions) and alternate_sign as false. The tokenizer function is set as the value for the tokenizer parameter.



```
vect = HashingVectorizer(decode_error='ignore', n_features=2**21,
                         preprocessor=None,tokenizer=preprocess_text,alternate_sign=False)
```

Figure 19: The Hashing Vectorizer

E. Train Test Split and Vectorization

The dataset is split 80% for training and 20% for test. After doing the split, the Hashing Vectorizer function is used to transform the text data into numeric data that will be used to train the classification models.

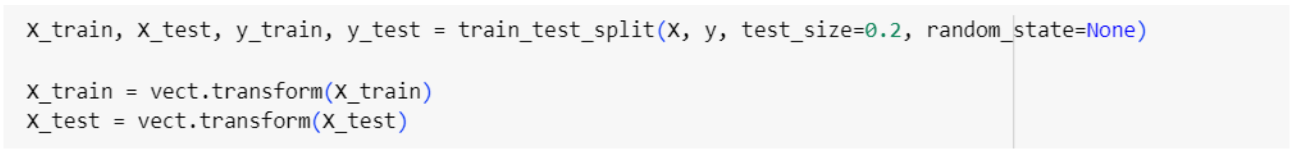

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=None)

X_train = vect.transform(X_train)
X_test = vect.transform(X_test)
```

Figure 20: Train Test Split and Vectorization

F. Performance Metrics and Evaluation

The models are evaluated using accuracy, precision, f1 score, and recall. The aforementioned metrics can be found by using values from the confusion matrix, which is part of the output of the evaluation. The confusion matrix that contains the actual values of the predictions would also be generated to assist with comparing the different models and evaluating which model would be most suitable for the output application.

G. Application Development

This study aims to construct a Discord bot, which is an automated program that runs specific tasks on the social media platform Discord. The bot is able to read messages being sent to the specific server and determine whether the message being sent contains suicidal intent or not. The identification is done by integrating the best performing model within the bot code and configuring the bot to display the user, message, and the probability result generated by the model. The use case diagram below demonstrates the roles of the intended users for the bot.

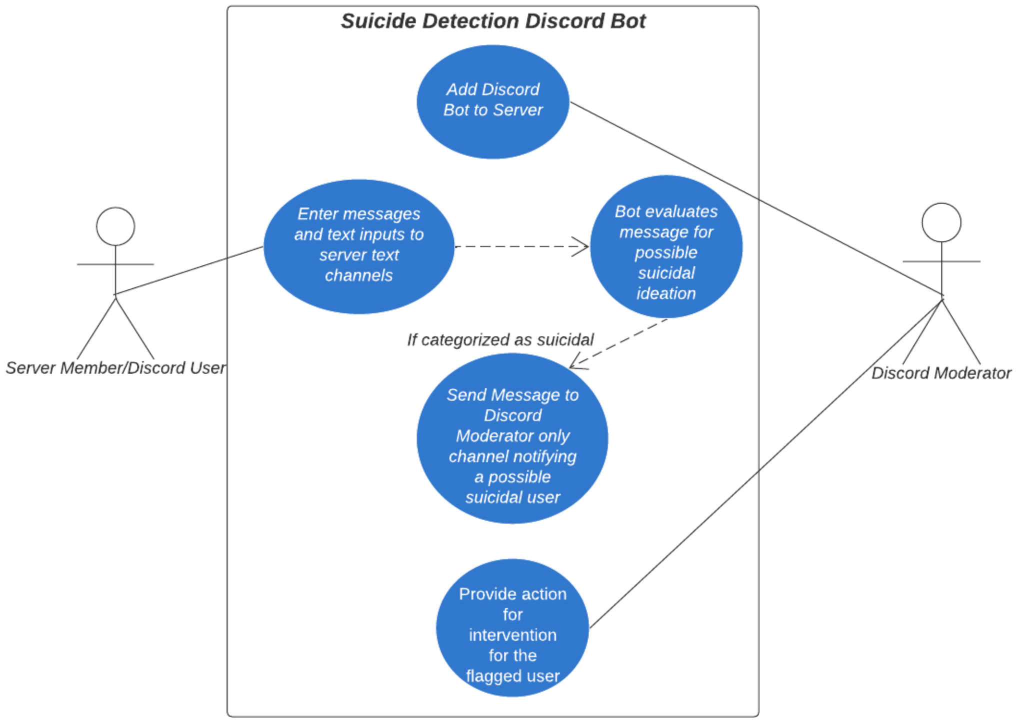
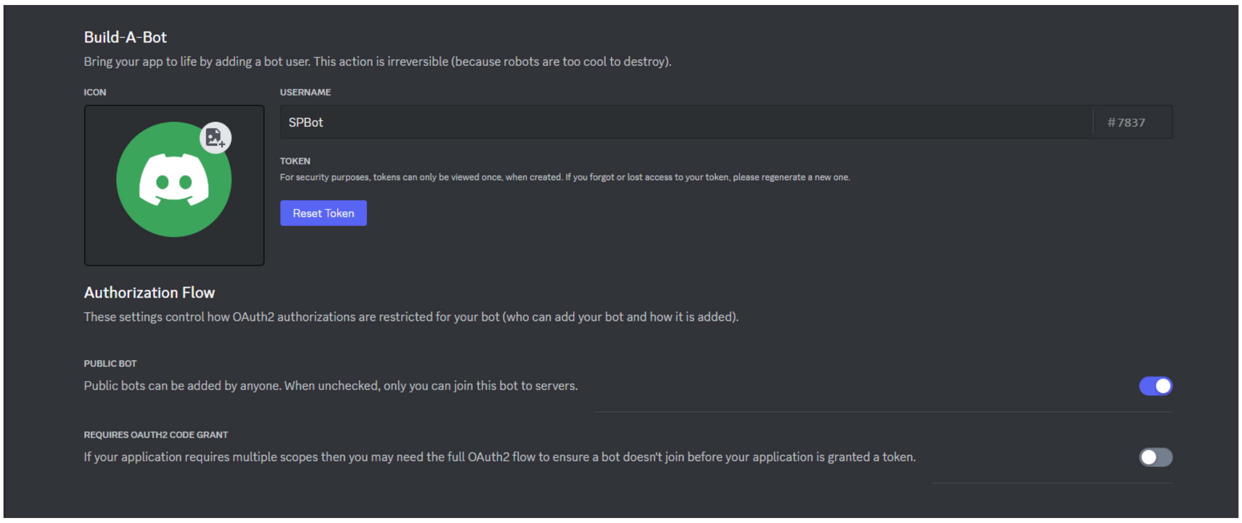

Suicide Detection Discord Bot
Add Discord Bot to Server
Enter messages and text inputs to server text channels
Bot evaluates message for possible suicidal ideation
If categorized as suicidal
Send Message to Discord Moderator only channel notifying a possible suicidal user
Provide action for intervention for the flagged user
Server Member/Discord User
Discord Moderator

Figure 21: Use Case Diagram

The creation of the bot is done through the Discord Developer Portal, which is a platform provided by Discord that allows developers to use their API to create automated programs that would assist Discord servers in all ways possible. This portal allows us to create a bot that also uses their own authentication for ease of development. Below is a screenshot of the bot creation window of the Discord Developer Portal.



Build-A-Bot

Bring your app to life by adding a bot user. This action is irreversible (because robots are too cool to destroy).

ICON

USERNAME

SPBot #7837

TOKEN

For security purposes, tokens can only be viewed once, when created. If you forgot or lost access to your token, please regenerate a new one.

Reset Token

Authorization Flow

These settings control how OAuth2 authorizations are restricted for your bot (who can add your bot and how it is added).

PUBLIC BOT

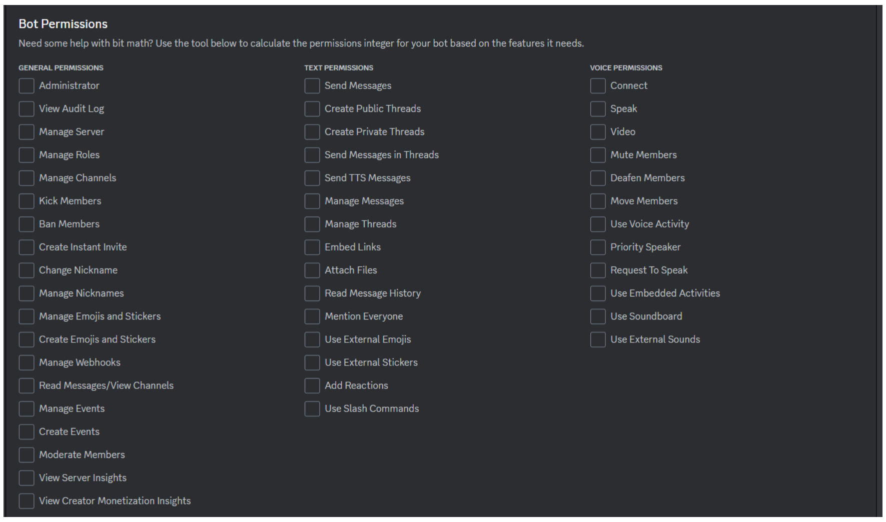
Public bots can be added by anyone. When unchecked, only you can join this bot to servers.

REQUIRES OAUTH2 CODE GRANT

If your application requires multiple scopes then you may need the full OAuth2 flow to ensure a bot doesn't join before your application is granted a token.

Figure 22: Discord Developer Portal Bot Creation

The bot creation feature provided by Discord also allows the developer to select the necessary permissions and actions that the bot must be able to execute to perform its intended purpose. This also addresses the need for consent among the future server owners and Discord users that would be using the bot. Below are the available features that can be enabled for the bot.



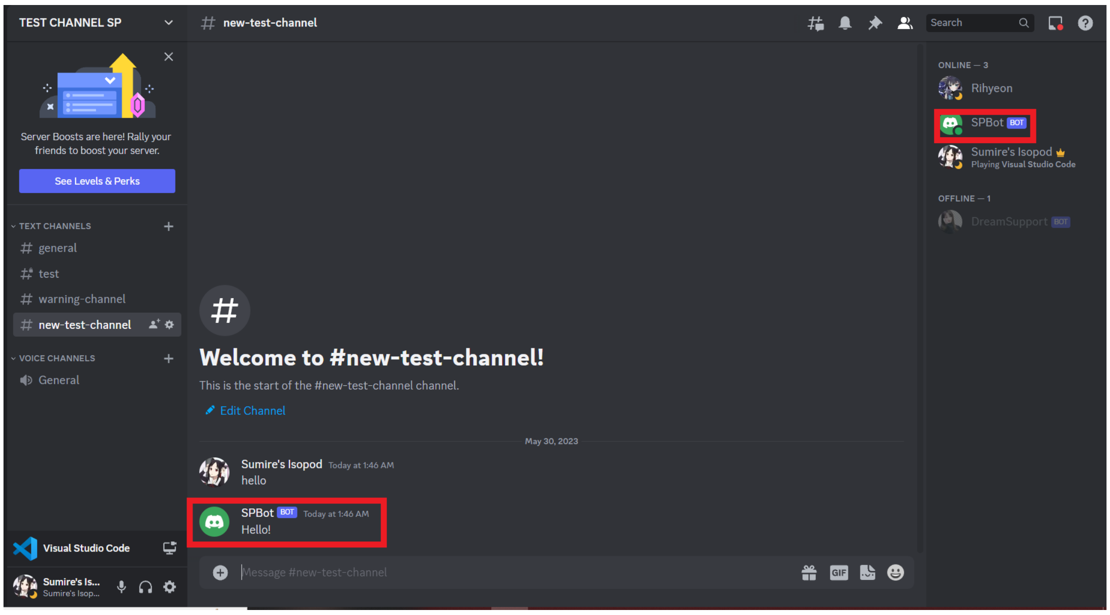
Bot Permissions

Need some help with bit math? Use the tool below to calculate the permissions integer for your bot based on the features it needs.

GENERAL PERMISSIONS	TEXT PERMISSIONS	VOICE PERMISSIONS
☐ Administrator	☐ Send Messages	☐ Connect
☐ View Audit Log	☐ Create Public Threads	☐ Speak
☐ Manage Server	☐ Create Private Threads	☐ Video
☐ Manage Roles	☐ Send Messages in Threads	☐ Mute Members
☐ Manage Channels	☐ Send TTS Messages	☐ Deafen Members
☐ Kick Members	☐ Manage Messages	☐ Move Members
☐ Ban Members	☐ Manage Threads	☐ Use Voice Activity
☐ Create Instant Invite	☐ Embed Links	☐ Priority Speaker
☐ Change Nickname	☐ Attach Files	☐ Request To Speak
☐ Manage Nicknames	☐ Read Message History	☐ Use Embedded Activities
☐ Manage Emojis and Stickers	☐ Mention Everyone	☐ Use Soundboard
☐ Create Emojis and Stickers	☐ Use External Emojis	☐ Use External Sounds
☐ Manage Webhooks	☐ Use External Stickers	
☐ Read Messages/View Channels	☐ Add Reactions	
☐ Manage Events	☐ Use Slash Commands	
☐ Create Events		
☐ Moderate Members		
☐ View Server Insights		
☐ View Creator Monetization Insights		

Figure 23: Bot Permissions for Discord

Finally, the bot can be invited to the server and can be coded for the suicide detection tool. The coding integrates the best performing model among the selection. Below is a screenshot of the test Discord server where I added the created bot.



Welcome to #new-test-channel!

This is the start of the #new-test-channel channel.

May 30, 2023

Sumire's Isopod Today at 1:46 AM
hello

SPBot BOT Today at 1:46 AM
Hello!

Figure 24: Bot Inside the Test Server

H. Programming Language

The data manipulation and model creation were developed primarily using Python's scikit-learn library. Python allows for access to versatile tools needed for manipulation of strings, data preprocessing, visualization of data, quantifying data, and statistical tools for model creation and evaluation.

The Logistic Regression is executed by using the LogisticRegression package from sklearn. linearmodel, the model is setup with the liblinear (Library for Large Linear Classification) solver to accomodate for the data dimensions. For Naive Bayes, the MultinomialNB package from sklearn.naive_bayes is utilized, setting the vectorizer to not accomodate negative values is crucial for the Naive Bayes package to work. Lastly, the Extreme Gradient Boosting is done by using the

XGBClassifier from xgboost as a separate python package. The XGBoost is used with eval_metric as error for binary classification.

The bot is developed by using Discord py, which is a Python library that exhaustively implements Discord's API. This is used to configure commands, receive and send messages, and to maintain bot functionality. Python is the code used for coding the functionalities of the entire bot.

V. Results

A. Machine Learning Models

Presented in this section is the result of training the machine learning models for binary classification of suicide and non-suicide based on text data. There are 3 models that are designed, trained, tested, and evaluated for performance metrics. The results are then analyzed and interpreted in order to determine if the NLP and preprocessing techniques applied to the text allowed the machine learning algorithms to produce a satisfactory performance that will be viable for real life applications. Comparison of evaluation results has also been done to decide whether Logistic Regression, Naive Bayes, or XGBoost would be the best performer for the data that we produced for model creation.

The section below would also show the confusion matrix along with the performance metrics results for each model, with a brief description of how it performed in general with our data. The data is split 80-20 for all models, amounting to 185,659 total training data and 46,415 testing data.

1. Logistic Regression

	Accuracy	Precision	F1 Score	Recall
Logistic Regression	93.24%	93.86%	93.20%	92.55%

Table 1: Logistic Regression Performance Metrics

Logistic Regression performed great, correctly identifying 43,276 entries out of a total of 46,415. The Logistic Regression classifier also correctly identified 21,502 out of 23,234 suicide entries, which is 92.55%. It also correctly identified 21,774 out of 23,181 non-suicide entries, which is 93.90%. While it performed slightly better at determining non-suicide entries, it is only by a very small margin and the performance for detecting both suicide and non-suicide entries are still relatively high. The confusion matrix for Logistic Regression is shown below.

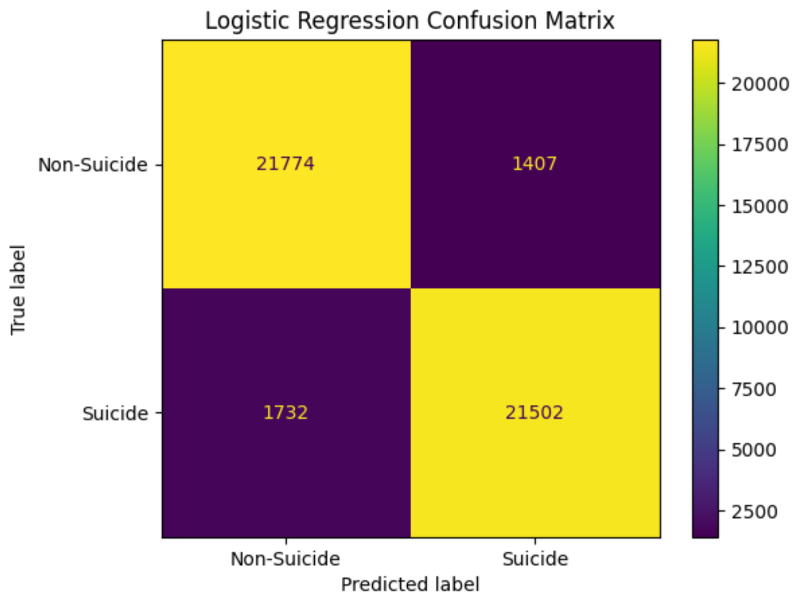


Figure 25: Logistic Regression Confusion Matrix

2. Naive Bayes

	Accuracy	Precision	F1 Score	Recall
Naive Bayes	76.24%	67.95%	80.74%	99.46%

Table 2: Naive Bayes Performance Metrics

Naive Bayes performed significantly worse than the previous model, only correctly identifying 35,388 out of the total 46,415 entries in the dataset. It is notable though that the Naive Bayes classifier has a significant advantage in determining suicide entries, being able to identify all but 126 suicide entries, thus having a very high recall score. However, the model showed relatively poor performance in detecting non-suicide entries. Only identifying 12,280 out of 23,181 entries which is only 53%. Despite the near-perfect performance in identifying suicide entries, the low performance on detecting non-suicide entries impacts the precision score of the model, which in turn drags down its total f1 score. The confusion matrix for Naive Bayes is shown below.

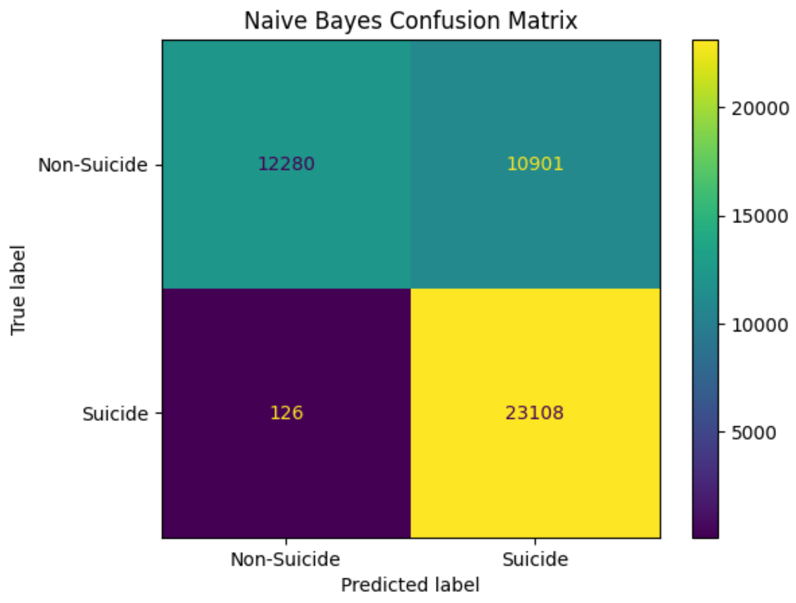


Figure 26: Naive Bayes Confusion Matrix

3. XGBoost

	Accuracy	Precision	F1 Score	Recall
XGBoost	91.27%	93.16%	91.09%	89.12%

Table 3: XGBoost Performance Metrics

XGBoost performed relatively well. It is able to identify 42,365 entries out of 46,415 correctly. Contrary to the Naive Bayes model, XGBoost performed better at identifying non-suicide entries, correctly identifying 21,660 entries out of 23,181 total non-suicide entries. While Naive Bayes is still significantly better at identifying suicide entries, XGBoost still performed relatively well in this regard, being able to identify 20,705 suicide entries out of 23,234. Despite performing worse in identifying suicide entries than Naive Bayes, the performance of the model in identifying non-suicide entries allowed it to maintain a significantly higher f1 score than Naive Bayes, but slightly worse than Logistic Regression. The confusion matrix for XGBoost is shown below.

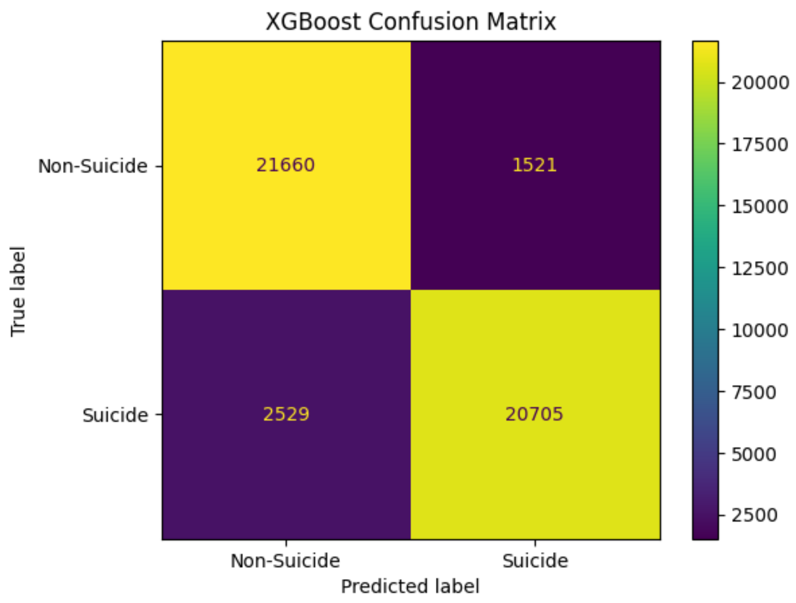


Figure 27: XGBoost Confusion Matrix

Below is the summary of all metric scores of all the models involved

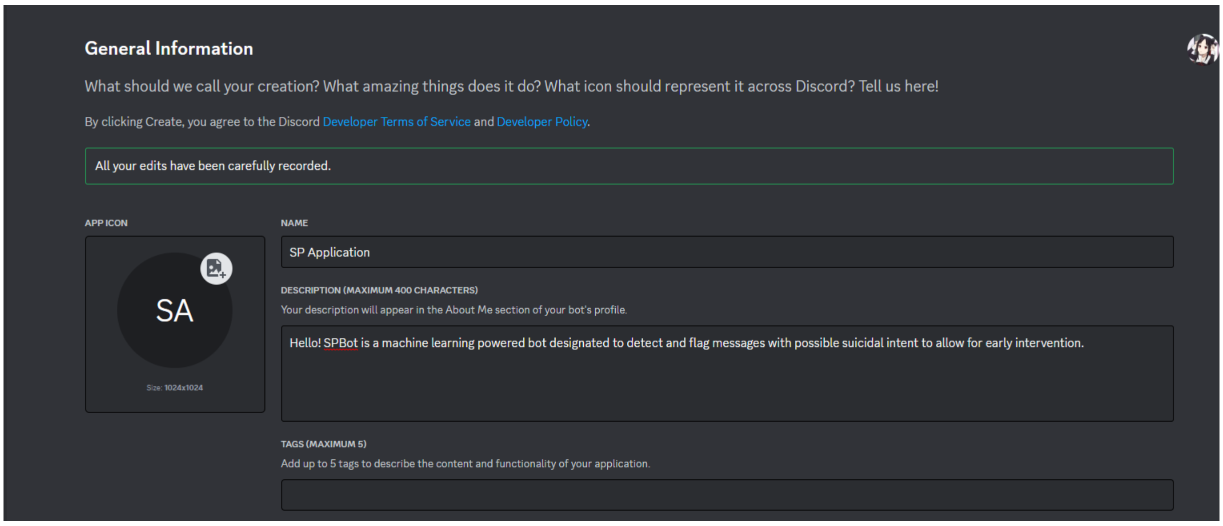
	Accuracy	Precision	F1 Score	Recall
Logistic Regression	93.24%	93.86%	93.20%	92.55%
Naive Bayes	76.24%	67.95%	80.74%	99.46%
XGBoost	91.27%	93.16%	91.09%	89.12%

Table 4: Summary of Metric Scores for All Models

B. Discord Application

1. Creating the Application and the Bot

   The way we approached the Discord Application is that we embed the model in a bot that would allow it to perform automated tasks in a Discord server. Therefore, creating the bot is the initial step to start the process. The Discord Developer portal provides the feature to create an application space, where we can generate the bot with the necessary permissions.


General Information

What should we call your creation? What amazing things does it do? What icon should represent it across Discord? Tell us here!

By clicking Create, you agree to the Discord Developer Terms of Service and Developer Policy.

All your edits have been carefully recorded.

APP ICON

SA

Size: 1024x1024

NAME

SP Application

DESCRIPTION (MAXIMUM 400 CHARACTERS)

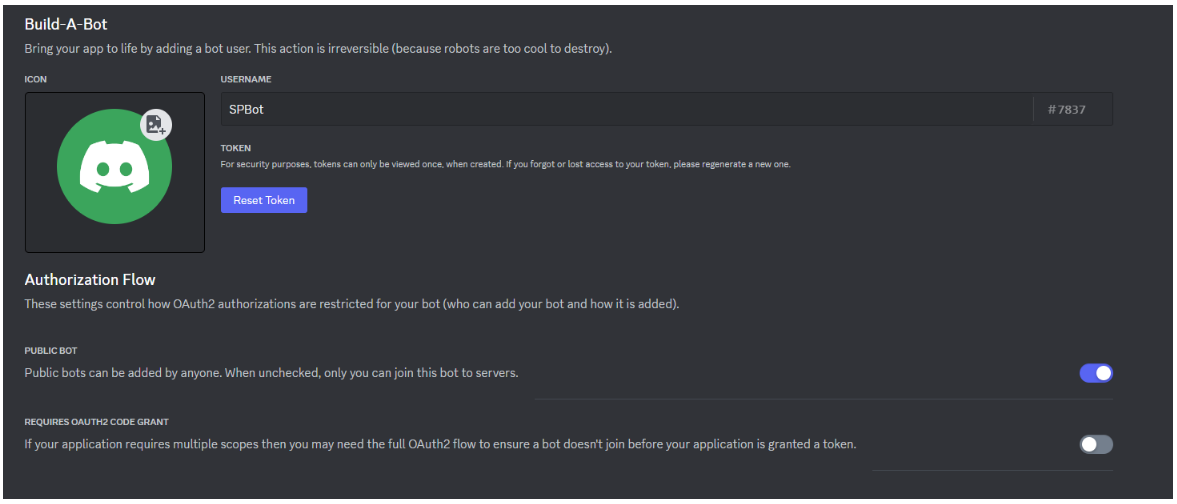
Your description will appear in the About Me section of your bot's profile.

Hello! SPBot is a machine learning powered bot designated to detect and flag messages with possible suicidal intent to allow for early intervention.

TAGS (MAXIMUM 5)

Add up to 5 tags to describe the content and functionality of your application.

Figure 28: Application Creation Page



Build-A-Bot

Bring your app to life by adding a bot user. This action is irreversible (because robots are too cool to destroy).

ICON

USERNAME

SPBot #7837

TOKEN

For security purposes, tokens can only be viewed once, when created. If you forgot or lost access to your token, please regenerate a new one.

Reset Token

Authorization Flow

These settings control how OAuth2 authorizations are restricted for your bot (who can add your bot and how it is added).

PUBLIC BOT

Public bots can be added by anyone. When unchecked, only you can join this bot to servers. [✓]

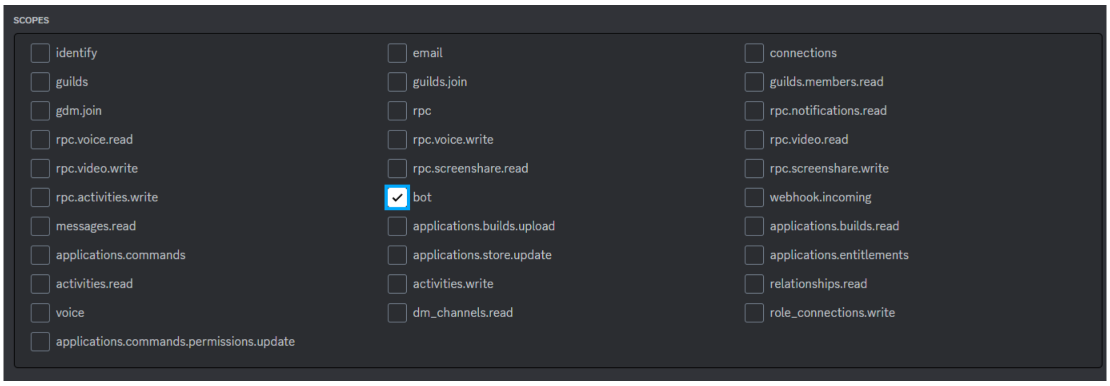
REQUIRES OAUTH2 CODE GRANT

If your application requires multiple scopes then you may need the full OAuth2 flow to ensure a bot doesn't join before your application is granted a token.

Figure 29: Bot Creation Page

2. Configuring the Bot Permissions

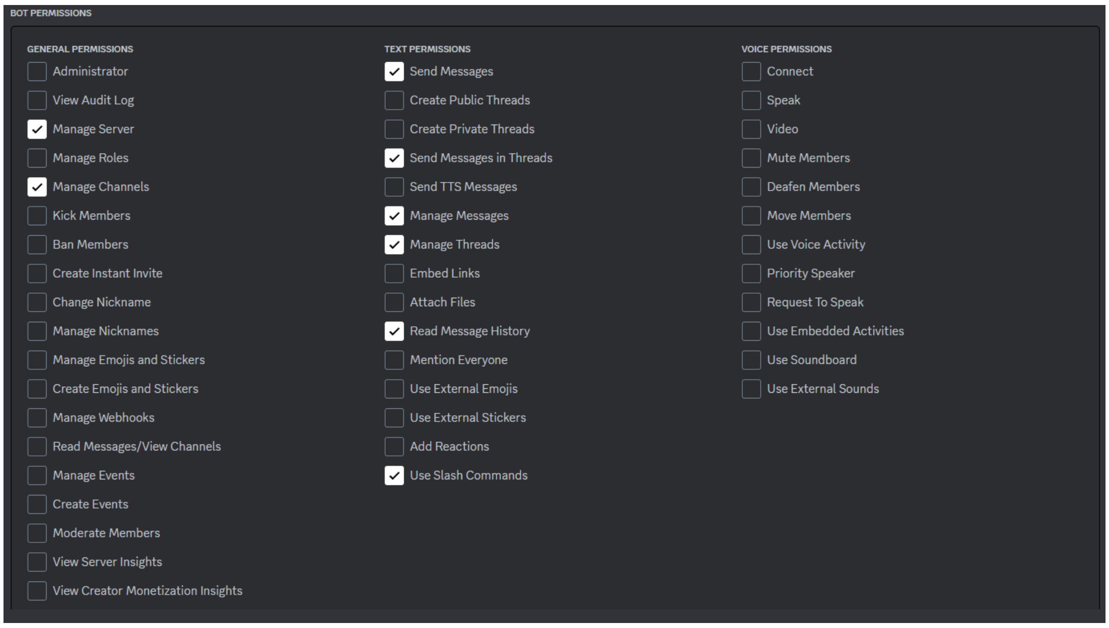
   This step allowed us to specify what specific permissions the bot needs in order to perform its functionalities. This would also display to the users the activities that the bot is able to perform within the server such as reading messages, managing channels, or joining voice calls. This allows the users to practice discretion when inviting specific bots to their servers. Our bot needed to be able to manage server and channels, as well as have permissions for all text related features in order to perform its functionalities well



SCOPES

☐ identify	☐ email	☐ connections
☐ guilds	☐ guilds.join	☐ guilds.members.read
☐ gdm.join	☐ rpc	☐ rpc.notifications.read
☐ rpc.voice.read	☐ rpc.voice.write	☐ rpc.video.read
☐ rpc.video.write	☐ rpc.screenshare.read	☐ rpc.screenshare.write
☐ rpc.activities.write	✓ bot	☐ webhook.incoming
☐ messages.read	☐ applications.builds.upload	☐ applications.builds.read
☐ applications.commands	☐ applications.store.update	☐ applications.entitlements
☐ activities.read	☐ activities.write	☐ relationships.read
☐ voice	☐ dm_channels.read	☐ role_connections.write
☐ applications.commands.permissions.update		

Figure 30: Scope Selection Page



BOT PERMISSIONS

GENERAL PERMISSIONS
- ☐ Administrator
- ☐ View Audit Log
- ☑ Manage Server
- ☐ Manage Roles
- ☑ Manage Channels
- ☐ Kick Members
- ☐ Ban Members
- ☐ Create Instant Invite
- ☐ Change Nickname
- ☐ Manage Nicknames
- ☐ Manage Emojis and Stickers
- ☐ Create Emojis and Stickers
- ☐ Manage Webhooks
- ☐ Read Messages/View Channels
- ☐ Manage Events
- ☐ Create Events
- ☐ Moderate Members
- ☐ View Server Insights
- ☐ View Creator Monetization Insights

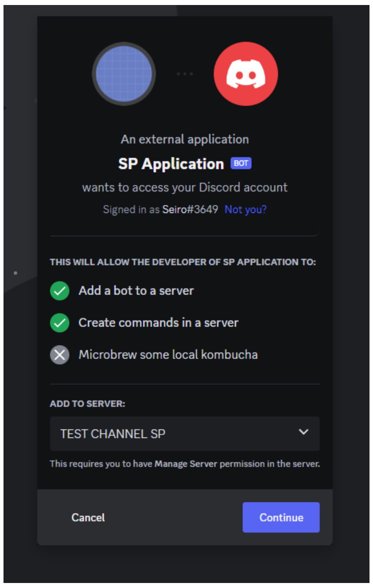
TEXT PERMISSIONS
- ☑ Send Messages
- ☐ Create Public Threads
- ☐ Create Private Threads
- ☑ Send Messages in Threads
- ☐ Send TTS Messages
- ☑ Manage Messages
- ☑ Manage Threads
- ☐ Embed Links
- ☐ Attach Files
- ☑ Read Message History
- ☐ Mention Everyone
- ☐ Use External Emojis
- ☐ Use External Stickers
- ☐ Add Reactions
- ☑ Use Slash Commands

VOICE PERMISSIONS
- ☐ Connect
- ☐ Speak
- ☐ Video
- ☐ Mute Members
- ☐ Deafen Members
- ☐ Move Members
- ☐ Use Voice Activity
- ☐ Priority Speaker
- ☐ Request To Speak
- ☐ Use Embedded Activities
- ☐ Use Soundboard
- ☐ Use External Sounds

Figure 31: Setting up Necessary Bot Permissions

3. Inviting the Bot to the Server

   After setting up the scope and permissions, Discord automatically generated a URL to allow us to select a server to add the bot into. Discord automatically popped up the servers that you have permissions to add members into, and we simply selected the server where the bot needs to join. After selection, an authorize page popped up notifying the administrator about the permissions the bot needs in order to operate in the server. Once authorized, the bot will then enter the server as another user.


An external application

SP Application BOT

wants to access your Discord account

Signed in as Seiro#3649 Not you?

THIS WILL ALLOW THE DEVELOPER OF SP APPLICATION TO:

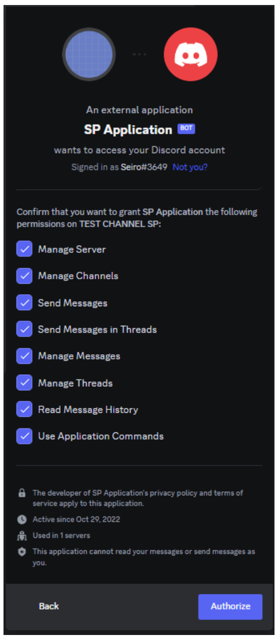
- ✓ Add a bot to a server
- ✓ Create commands in a server
- ✗ Microbrew some local kombucha

ADD TO SERVER:

TEST CHANNEL SP

This requires you to have Manage Server permission in the server.

Figure 32: Generated URL Page: Server Selection Screen


An external application

SP Application BOT

wants to access your Discord account

Signed in as Seiro#3649 Not you?

Confirm that you want to grant SP Application the following permissions on TEST CHANNEL SP:

- [✓] Manage Server
- [✓] Manage Channels
- [✓] Send Messages
- [✓] Send Messages in Threads
- [✓] Manage Messages
- [✓] Manage Threads
- [✓] Read Message History
- [✓] Use Application Commands

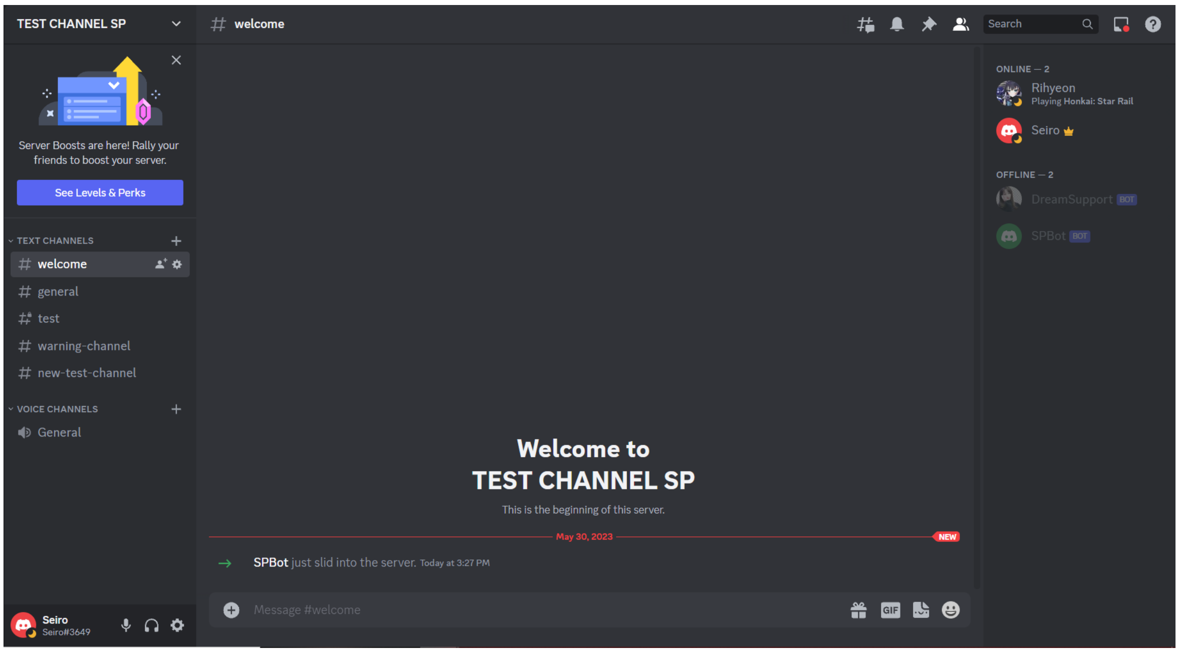
The developer of SP Application's privacy policy and terms of service apply to this application.

Active since Oct 29, 2022

Used in 1 servers

This application cannot read your messages or send messages as you.

Figure 33: Generated URL Page: Server Authorization



Welcome to
TEST CHANNEL SP
This is the beginning of this server.

May 30, 2023

SPBot just slid into the server. Today at 3:27 PM

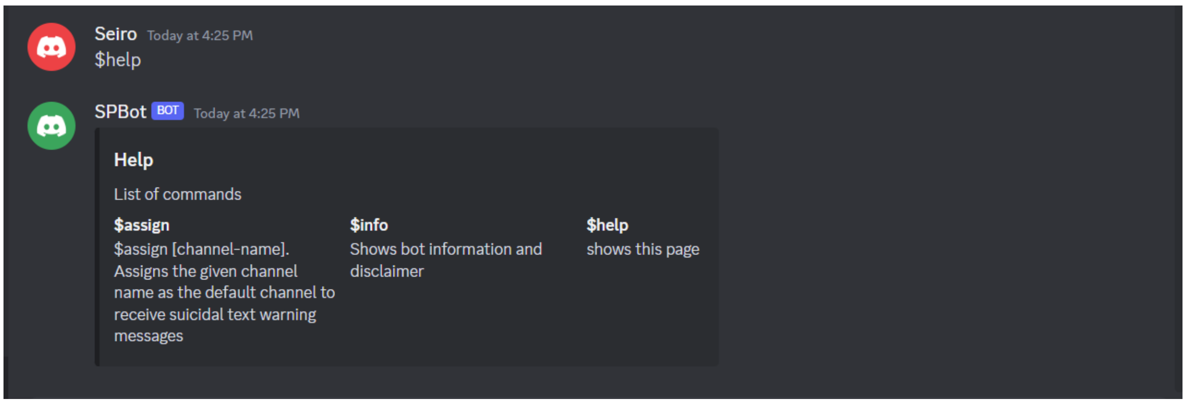
Figure 34: The Server Welcome Message after Bot Joins

4. Implementing the bot features

   This section will individually describe the bot features

   (a) The help command

       This is a standard command to display the commands and features available for user input



Seiro Today at 4:25 PM
$help

SPBot BOT Today at 4:25 PM

Help

List of commands

$assign
$assign [channel-name]. Assigns the given channel name as the default channel to receive suicidal text warning messages

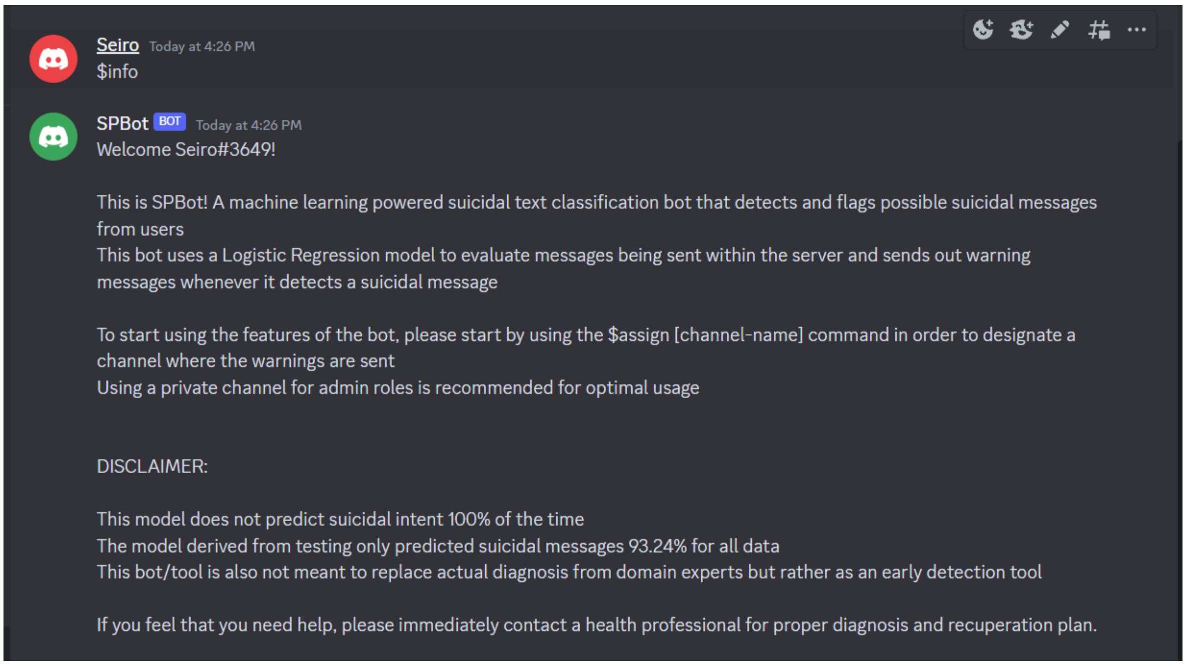
$info
Shows bot information and disclaimer

$help
shows this page

Figure 35: Help Command

(b) The info command

A command that displays to the user a brief description about the bot



Seiro Today at 4:26 PM
$info

SPBot BOT Today at 4:26 PM
Welcome Seiro#3649!

This is SPBot! A machine learning powered suicidal text classification bot that detects and flags possible suicidal messages from users
This bot uses a Logistic Regression model to evaluate messages being sent within the server and sends out warning messages whenever it detects a suicidal message

To start using the features of the bot, please start by using the $assign [channel-name] command in order to designate a channel where the warnings are sent
Using a private channel for admin roles is recommended for optimal usage

DISCLAIMER:

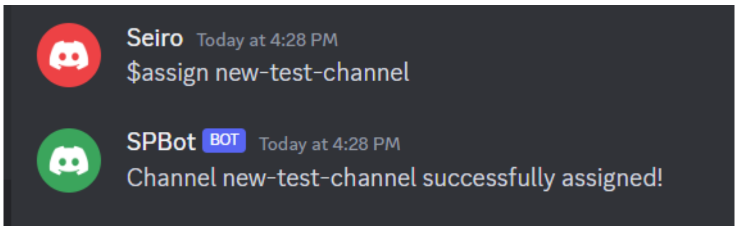
This model does not predict suicidal intent 100% of the time
The model derived from testing only predicted suicidal messages 93.24% for all data
This bot/tool is also not meant to replace actual diagnosis from domain experts but rather as an early detection tool

If you feel that you need help, please immediately contact a health professional for proper diagnosis and recuperation plan.

Figure 36: Info Command

(c) The assign command

A command that accepts a currently existing text channel within the server as a parameter. Sets this channel as the default channel to receive the suicidal warning messages. This is set as a server owner only accessible channel for safeguard of usage. Any member that is not the server owner will not be able to reassign channels.

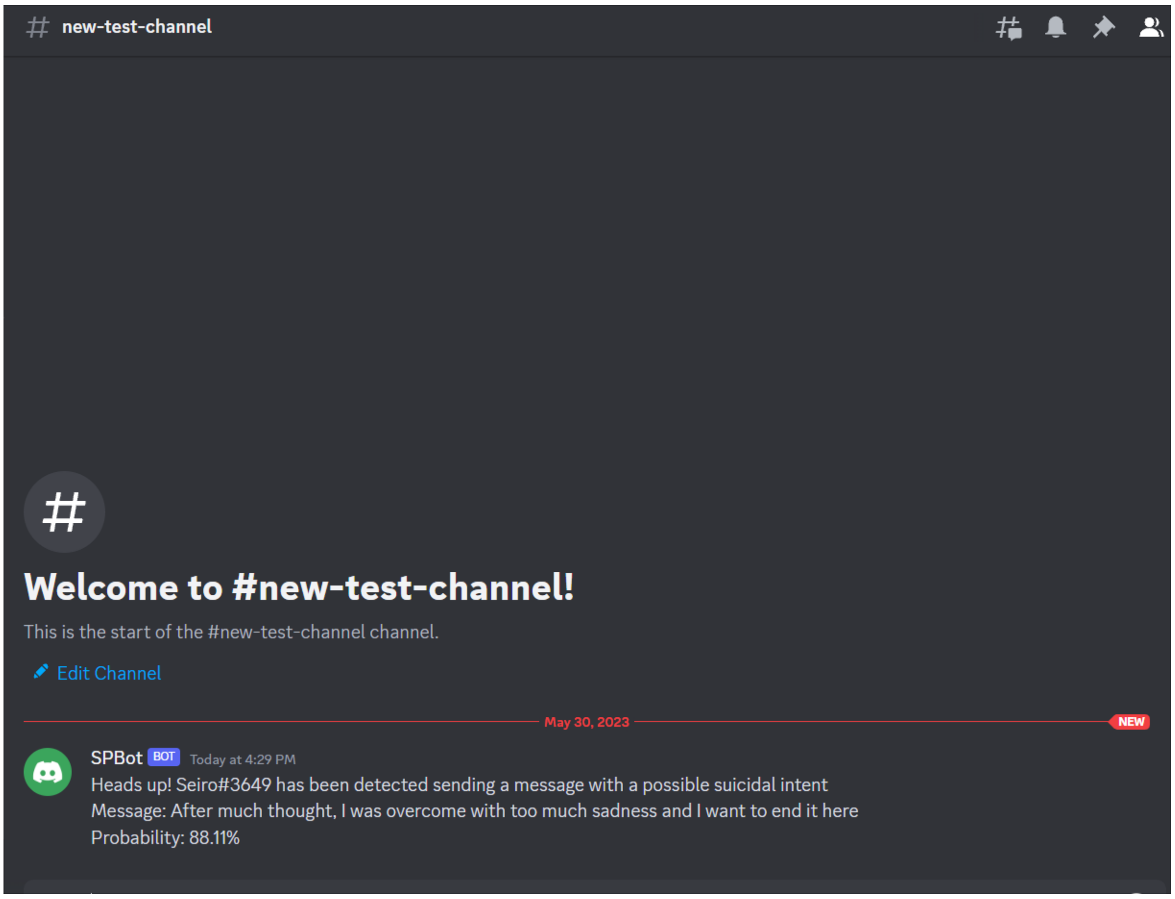


Seiro Today at 4:28 PM
$assign new-test-channel

SPBot BOT Today at 4:28 PM
Channel new-test-channel successfully assigned!

Figure 37: Assign Command

When there is a currently assigned channel, the bot will then start to automatically track the messages being sent within the server, and will send warning messages accordingly, if the model predicts a possible suicidal intent message. The warning message will contain the user that sent the message, the message itself, and the probability in which the classifier predicts that it might be suicidal.



new-test-channel
Welcome to #new-test-channel!
This is the start of the #new-test-channel channel.
Edit Channel
May 30, 2023
NEW
SPBot BOT Today at 4:29 PM

Heads up! Seiro#3649 has been detected sending a message with a possible suicidal intent
Message: After much thought, I was overcome with too much sadness and I want to end it here
Probability: 88.11%

Figure 38: Sample Bot Output

VI. Discussions

The suicide detection dataset is nothing short of a challenge for the researcher. The large amount of data that it contained presented itself well for the researcher to apply numerous preprocessing techniques in order to only extract the powerful predicting features, in this case the important words, that would be impactful in the model performance. This dataset also reflected the challenge of working with social media extracted data. The informal language of social media, along with the use of emoticons and sudden punctuations, created a lot of inconsistency within the data. It is noted during the preprocessing steps that the volume of data was significantly reduced after removal of emoticons, punctuations, non-english words, and stop words.

The study also compared three machine learning models for predicting the entries. Logistic Regression, Naive Bayes, and Extreme Gradient Boosting (XGBoost) are all fed with a vectorized text data using the Hashing Vectorizer function. Logistic Regression came out as the best model. Logistic Regression works well with sparse matrices, which is what the Hashing Vectorizer produces. The output variable is also binary, which is a very common use for Logistic Regression since it performs well with binary classification problems. Naive Bayes performed relatively poor among its other competitors in this comparison. Based on the wordcloud, various different words are appearing frequently which is different from both the suicide and non-suicide entries, which meant that there is a high probability that the training data would have a disparity with the test data in terms of the frequency distribution. Non-representation of this might be a factor why the Naive Bayes model performed inferior to the others. Also, Naive Bayes is good as a classifier but bad as an estimator. So in terms of the probability value, the probability output from the Logsitic Regression model is more reliable. However, Naive Bayes did earn the highest recall score, but at the cost of a fall off in precision score, which signifies a large amount of false positive tagging. XGBoost also performed well with the dataset. It performed slightly worse than Logistic

Regression, but it also obtained high metric scores for a faster running time. XG-Boost's boosting ability and also its efficient handling of missing values makes it a valuable alternative to Logistic Regression, and might even perform better with a higher quality dataset.

The model despite performing well within the dataset, is not perfect. During the testing phase of the application, there are some common themes and textual patterns observed that made the model underpeform, or produce a wrong prediction. For instance, the model did not take into account perspectives in speech, which makes topic about suicide, even though not about the actual person sending the message, can be tagged as suicidal for that person. Talking about a form of media, like a movie, with content that are related to suicide would also be tagged as suicidal for the sender, despite not being supposed to be flagged. On the other hand, there are also scenarios which some suicidal texts are not being identified. These occured most commonly on expressions, and niche language that are not generally used for common speech. Expressions such as "kicking the bucket" or "six feet under", are not being identified because the words themselves individually are not closely related to suicide. Addressing these problems would require new approaches. For the false positive problem, additional preprocessing steps and nlp techniques can be explored to consider speaker perspective, or sentiment analysis. For the false negative problem, this is more reliant on the data itself. Data with more language adaptations, or data from a certain niche or subset of people that might be exposed to some language nuances will put weight to some of the more cryptic expressions. However, this must be done in a way that will not compromise the classification using common words, or not to put too much weight on its individual words, as putting too much weight on the words themselves, can be the cause for another false positive problem.

Creating the bot has been quite an interesting challenge. After the models were saved, the first issue was a version conflict with my local version of Python packages. After updating my sklearn package, the model was then loaded and is ready to be embedded in the bot. The actual creation of the bot and inviting it to the server is a relatively straightforward task since they are features that are provided by Discord directly. Working with the API is where the development becomes a bit challenging. One of the challenges encountered is that, the bot reads it own messages as well, since the bot sends the warning message along with the text itself, it flags its own warning message as suicidal and then it loops infinitely until the program crashes. It was worked around by filtering out bot messages on the on_message event. Configuration of the help page was also a bit of a challenge since Discord has a default help constructed for each bot. The method done for this is to remove the default help command provided by the bot package and override it with a custom-made help function, which also allowed us to improve more on its formatting.

VII. Conclusions

To conclude, the dataset was preprocessed using various NLP and preprocessing techniques such as tokenization, stemming, removal of stop words, removal of non-english words, removal of emojis, label-encoding, and converted to quantifiable vectorized data using Hashing Vectorizer, and is used by three classification machine learning models which are Logistic Regression, Naive Bayes, and XGBoost. Logistic Regression performed the best among all of the three models with metric scores of 93.24% accuracy score, 93.86% precision score, 93.20% F1 score, and 92.55% recall score. Following closely second is the XGBoost model, and taking the third and last place by a significant margin would be the Naive Bayes model.

It is shown the preprocessing heavily impacted performance and scalability of the methods. Not only did preprocessing remove noise, but it also formatted the data that is easily interpretable by the models. However, choosing nlp and preprocessing techniques is an area where there is still a vast potential of improvement and other preprocessing steps can allow different models to shine depending on how the data is quantified. As we have seen in the results, despite the three classification techniques performing considerably well in general for text classification problems, data representation can significantly impact a model's performance like Naive Bayes in this study.

The Discord bot is currently run locally. It reads every message sent to the server it is a member of and it can detect and flag messages that have a suicidal tendency. A warning message will then be sent to a text channel of the user's choice. It can also run some additional commands such as help for instructions on how the bot works, and info for a brief description of the bot and its functionalities.

VIII. Recommendations

1. Dataset Exploration

   The dataset is vast and can be used for a multitude of other applications aside the suicide and non-suicide classifier. The dataset can be explored further for sentiment analysis, and also for finding reoccuring topics between suicide and non-suicide social media posts.

2. Feature Extraction/Selection

   With various techniques further emerging from NLP, there are other alternatives that can be used for the dataset, such as word2vec, LDA, CountVectorizer, etc. These tools can be used to further enhance the data quality, or to represent the data in a different way that can allow other models to perform good predictions.

3. Other Possible Datasets

   The study that is performed is still quite limited in terms of scope, and also cannot capture the entire sentimentality of the social media post. Future study in this field should consider working with datasets with other languages, since social media is something that can be accessed worldwide. Other improvements would also take the emoticons into account (as these are also expression of emotion for the person), and also analyzing image and other multimedia messages.

4. Other Applications

   The main application for this study is the Discord bot, which was made possible because of Discord's API which allows for developers to create bots for server use. However, the application of models like these can also be expanded for other social media platforms. These can also be implemented in a web application for predicting suicidal text in forums, blogs, or other forms of website. It is also possible to implement this as a stand alone application that can assist domain experts in early suicide intent detection. These are just some of the other applications that can be derived from this project.

IX. Bibliography

[1] K. C. Bathina, M. ten Thij, L. Lorenzo-Luaces, L. A. Rutter, and J. Bollen, “Depressed individuals express more distorted thinking on social media.,” *arXiv: Social and Information Networks*, 2 2020.

[2] E. A. Swedo, J. L. Beauregard, S. de Fijter, L. Werhan, K. Norris, M. P. Montgomery, E. B. Rose, C. David-Ferdon, G. M. Massetti, S. D. Hillis, and S. A. Sumner, “Associations Between Social Media and Suicidal Behaviors During a Youth Suicide Cluster in Ohio,” *Journal of Adolescent Health*, vol. 68, pp. 308–316, 2 2021.

[3] T. Miyagi, N. Nawa, P. J. Surkan, and T. Fujiwara, “Social media monitoring of suicidal content and change in trends of Japanese Twitter content around the Zama Suicide Pact Slayings,” *Psychiatry Research-neuroimaging*, p. 114490, 2 2022.

[4] S. Ji, S. Pan, X. Li, E. Cambria, G. Long, and Z. Huang, “Suicidal Ideation Detection: A Review of Machine Learning Methods and Applications,” *IEEE Transactions on Computational Social Systems*, vol. 8, pp. 214–226, 2 2021.

[5] M. Chatterjee, P. Kumar, P. Samanta, and D. Sarkar, “Suicide ideation detection from online social media: A multi-modal feature based technique,” *International Journal of Information Management Data Insights*, 11 2022.

[6] X. Wang, C. Wang, J. Yao, H. Fan, Q. Wang, Y. Ren, and Q. Gao, “Comparisons of deep learning and machine learning while using text mining methods to identify suicide attempts of patients with mood disorders,” *Journal of Affective Disorders*, vol. 317, pp. 107–113, 11 2022.

[7] N. Nordin, Z. Zainol, M. H. M. Noor, and L. F. Chan, “An Explainable Predictive Model for Suicide Attempt Risk using An Ensemble Learning and Shapley Additive Explanations (SHAP) Approach,” *Asian Journal of Psychiatry*, p. 103316, 11 2022.

[8] E.-L. Wu, C.-Y. Wu, M.-B. Lee, K.-C. Chu, and M.-S. Huang, "Development of Internet suicide message identification and the Monitoring-Tracking-Rescuing model in Taiwan," *Journal of Affective Disorders*, vol. 320, pp. 37–41, 1 2023.

[9] S. M. Sarsam, H. Al-Samarraie, A. I. Alzahrani, W. Alnumay, and A. P. Smith, "A lexicon-based approach to detecting suicide-related messages on Twitter," *Biomedical Signal Processing and Control*, vol. 65, p. 102355, 3 2021.

[10] D. Morrow, R. Zamora-Resendiz, J. C. Beckham, N. A. Kimbrel, D. W. Oslin, S. Tamang, and S. Crivelli, "A case for developing domain-specific vocabularies for extracting suicide factors from healthcare notes," *Journal of Psychiatric Research*, vol. 151, pp. 328–338, 2022.

X. Appendix

A. Source Code

```
#include <iostream>

using namespace std;

int main{
        cout << "Hello world!" << endl;
        return 0;
}
```

```
# -*- coding: utf-8 -*-
"""sp.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/16B9athQmuShzDV_nVPf9j55KbuCXqmL7

**IMPORT DEPENDENCIES**
"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn import preprocessing
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,accuracy_score,precision_score,f1_score,re
from tqdm import tqdm
import nltk
import re
import pickle
import torch
import string

nltk.download('words')
nltk.download('stopwords')

from nltk.stem.porter import PorterStemmer
from nltk.corpus import stopwords

from sklearn.feature_extraction.text import HashingVectorizer

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

from wordcloud import WordCloud, STOPWORDS

"""**IMPORT DATASET**"""

tqdm.pandas()
df = pd.read_csv('/content/drive/MyDrive/SP/Suicide_Detection.csv', engine='python', encoding='utf-8', on_bad_l
# display(df)
print(df.count())

"""**SAMPLE CORPUS**"""

corpus = [
     'This is the first document.',
     'This document is the second document.',
     'And this is the third one.',
     'Is this the first document?',
  ]

vectorizer = HashingVectorizer(n_features=2**4)
X = vectorizer.fit_transform(corpus)
print(X, '\n')
print(X.shape, '\n')

"""**EXPLORATORY DATA ANALYSIS**"""

df.head()
df.drop(columns=["Unnamed: 0"])
df.count()
df_suicide = len(df[df["class"]=="suicide"])
df_non_suicide = len(df[df["class"]=="non-suicide"])

print(df.head())
print("Total data points: " + "\n" + str(df.count()) + "\n")
print("Total data points for suicide: " + str(df_suicide))
print("Total data points for non-suicide: " + str(df_non_suicide))

# HISTOGRAMS
# df['text'].str.split().map(lambda x: len(x)).hist()

df_non_suicide= df[df['class']=='non-suicide']
df_non_suicide['text'].str.split().map(lambda x: len(x)).hist()

df_suicide= df[df['class']=='suicide']
```

```
df_suicide['text'].str.split().map(lambda x: len(x)).hist()

df['text'].str.split().map(lambda x: len(x)).hist()

"""df['text'].str.split().map(lambda x: len(x)).hist(range=[8000,10000])

**TEXT SPLITTING (RAW)**
"""

stop = stopwords.words('english')

corpus=[]
new= df['text'].str.split()
new=new.values.tolist()
corpus=[word for i in new for word in i]
corpus = list(filter(None, corpus))

print("Total number of words: " + str(len(corpus)))
print("Total number of distinct words: " + str(len(set(corpus))))

"""**TEXT SPLITTING (WITH PROCESSING)**"""

corpus_clean = []

regrex_pattern = re.compile(pattern = "["
        u"\U0001F600-\U0001F64F"  # emoticons
        u"\U0001F300-\U0001F5FF"  # symbols & pictographs
        u"\U0001F680-\U0001F6FF"  # transport & map symbols
        u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           "]+", flags = re.UNICODE)

for i in corpus:
  clean_text = regrex_pattern.sub(r'',i)
  no_punc_clean_text = clean_text.translate (str.maketrans ('', '', string.punctuation))
  no_punc_clean_text = no_punc_clean_text.replace("'", "")
  corpus_clean.append(no_punc_clean_text.lower())

corpus_clean = list(filter(None, corpus_clean))
words = set(nltk.corpus.words.words())
clean_text = []

for word in corpus_clean:
    if word in words and word not in stop:
      clean_text.append(word)

print("Total number of words: " + str(len(clean_text)))
print("Total number of distinct words after processing: " + str(len(set(clean_text))))

"""**WORDCLOUD**"""

# WORDCLOUD

stop_words = set(STOPWORDS)

text = " ".join(cat for cat in clean_text)

wordcloud = WordCloud(stopwords=stop_words, max_words=200)
wordcloud.generate(text)

plt.imshow(wordcloud,  interpolation='bilinear')
plt.axis("off")
plt.show()

"""**WORD COUNT**"""

# from collections import Counter
# import operator
# import itertools

# counts = Counter(clean_text)
# counts_sorted = dict( sorted(counts.items(), key=operator.itemgetter(1),reverse=True))

# sorted_data = pd.DataFrame(counts_sorted,index=[0])
# # print(sorted_data.head())
# top_50 = sorted_data.value_counts()[:50]
# print(top_50)



# top_50.plot(kind='bar',figsize=(10,8))
# plt.title('Top 50 words occuring in dataset')

"""**WORDCLOUD FOR SUICIDAL TEXT**"""

# df_suicide= df[df['class']=='suicide']
# new = df_suicide['text'].str.split()
# new=new.values.tolist()
# corpus_suicide =[word for i in new for word in i]
# corpus_suicide = list(filter(None, corpus_suicide))

# print("Total number of words: " + str(len(corpus_suicide)))
# print("Total number of distinct words: " + str(len(set(corpus_suicide))))

# corpus_clean = []

# regrex_pattern = re.compile(pattern = "["
#         u"\U0001F600-\U0001F64F"  # emoticons
#         u"\U0001F300-\U0001F5FF"  # symbols & pictographs
#         u"\U0001F680-\U0001F6FF"  # transport & map symbols
#         u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
#                            "]+", flags = re.UNICODE)

# for i in corpus_suicide:
#   clean_text = regrex_pattern.sub(r'',i)
#   no_punc_clean_text = clean_text.translate (str.maketrans ('', '', string.punctuation))
#   no_punc_clean_text = no_punc_clean_text.replace("'", "")
#   corpus_clean.append(no_punc_clean_text.lower())

# corpus_clean = list(filter(None, corpus_clean))
# words = set(nltk.corpus.words.words())
# clean_text = []

# for word in corpus_clean:
#     if word in words and word not in stop:
#       clean_text.append(word)

# # print("Total number of words after processing: " + str(len(corpus_clean)))
# print("Total number of distinct words after processing: " + str(len(set(clean_text))))

# text = " ".join(cat for cat in clean_text)

# wordcloud = WordCloud(stopwords=stop_words, max_words=200)
# wordcloud.generate(text)

# plt.imshow(wordcloud,  interpolation='bilinear')
# plt.axis("off")
# plt.show()

"""**WORDCLOUD FOR NON-SUICIDAL TEXT**"""

# df_non_suicide= df[df['class']=='non-suicide']
# new = df_non_suicide['text'].str.split()
# new=new.values.tolist()
# corpus_non_suicide =[word for i in new for word in i]
# corpus_non_suicide = list(filter(None, corpus_non_suicide))

# print("Total number of words: " + str(len(corpus_non_suicide)))
# print("Total number of distinct words: " + str(len(set(corpus_non_suicide))))

# corpus_clean = []

# regrex_pattern = re.compile(pattern = "["
#         u"\U0001F600-\U0001F64F"  # emoticons
#         u"\U0001F300-\U0001F5FF"  # symbols & pictographs
#         u"\U0001F680-\U0001F6FF"  # transport & map symbols
#         u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
#                            "]+", flags = re.UNICODE)

# for i in corpus_non_suicide:
#   clean_text = regrex_pattern.sub(r'',i)
#   no_punc_clean_text = clean_text.translate (str.maketrans ('', '', string.punctuation))
#   no_punc_clean_text = no_punc_clean_text.replace("'", "")
#   corpus_clean.append(no_punc_clean_text.lower())

# corpus_clean = list(filter(None, corpus_clean))
# words = set(nltk.corpus.words.words())
# clean_text = []

# for word in corpus_clean:
#     if word in words and word not in stop:
#       clean_text.append(word)

# # print("Total number of words after processing: " + str(len(corpus_clean)))
# print("Total number of distinct words after processing: " + str(len(set(clean_text))))

# text = " ".join(cat for cat in clean_text)

# wordcloud = WordCloud(stopwords=stop_words, max_words=200)
# wordcloud.generate(text)

# plt.imshow(wordcloud,  interpolation='bilinear')
# plt.axis("off")
# plt.show()

"""**PREPROCESSING AND NLP**"""

def preprocess_text(text):

    porter = PorterStemmer()

    def word_check(word):
        word_processed = porter.stem(word)
        word_processed="".join(letter for letter in word if letter.isalpha())
        return word_processed

    regrex_pattern = re.compile(pattern = "["
        u"\U0001F600-\U0001F64F"  # emoticons
        u"\U0001F300-\U0001F5FF"  # symbols & pictographs
        u"\U0001F680-\U0001F6FF"  # transport & map symbols
        u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           "]+", flags = re.UNICODE)


    stop = stopwords.words('english')

    text = regrex_pattern.sub(r'',text)
    text = text.lower()
    return [word_check(word) for word in text.split() if word not in stop]
```

```
# df['text'] = df['text'].progress_apply(preprocess_text)
# print(df)
# df.to_csv('Suicide_Detection_Preprocessed.csv')

df["class"] = df.apply(lambda x: 1 if x['class'] == 'suicide' else 0, axis=1)

# display(df)

vect = HashingVectorizer(decode_error='ignore', n_features=2**21,
                         preprocessor=None,tokenizer=preprocess_text,alternate_sign=False)

X = df["text"].to_list()
y = df['class']

"""**TRAIN-TEST SPLIT**"""

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

X_train = vect.transform(X_train)
X_test = vect.transform(X_test)

"""**LOGISTIC REGRESSION**"""

logreg = LogisticRegression(solver='liblinear')
logreg.fit(X_train, y_train)

y_pred = logreg.predict(X_test)

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Non-Suicide', 'Suicide'])
disp.plot()
disp.ax_.set_title("Logistic Regression Confusion Matrix")
plt.show()

ac = accuracy_score(y_test,y_pred)
precision = precision_score(y_test,y_pred)
f1_score_result = f1_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)


print("LOGREG \n")
print('Accuracy: %.2f%%' % (ac*100))
print('Precision: %.2f%%' % (precision*100))
print('F1 Score: %.2f%%' % (f1_score_result*100))
print('Recall: %.2f%%' % (recall*100))
print("======================")

model_save_name = 'classifier-LOGREG.pt'
path = F"/content/drive/MyDrive/SP/{model_save_name}"
torch.save(logreg, path)

"""**NAIVE-BAYES**"""

from sklearn.naive_bayes import MultinomialNB


classifier = MultinomialNB()
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,  display_labels=['Non-Suicide', 'Suicide'])
disp.plot()
disp.ax_.set_title("Naive Bayes Confusion Matrix")
plt.show()

ac = accuracy_score(y_test,y_pred)
precision = precision_score(y_test,y_pred)
f1_score_result = f1_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

ac = classifier.score(X_test, y_test)
precision = precision_score(y_test,y_pred)
f1_score_result = f1_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
print("NAIVE BAYES \n")
print('Accuracy: %.2f%%' % (ac*100))
print('Precision: %.2f%%' % (precision*100))
print('F1 Score: %.2f%%' % (f1_score_result*100))
print('Recall: %.2f%%' % (recall*100))
print("======================")

model_save_name = 'classifier-NB.pt'
path = F"/content/drive/MyDrive/SP/{model_save_name}"
torch.save(classifier, path)

"""**XGBOOST**"""

from xgboost import XGBClassifier



model = XGBClassifier(eval_metric='error')
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

```
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,  display_labels=['Non-Suicide', 'Suicide'])
disp.plot()
disp.ax_.set_title("XGBoost Confusion Matrix")
plt.show()

ac = accuracy_score(y_test, y_pred)
precision = precision_score(y_test,y_pred)
f1_score_result = f1_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
print("XGBOOST \n")
print('Accuracy: %.2f%%' % (ac*100))
print('Precision: %.2f%%' % (precision*100))
print('F1 Score: %.2f%%' % (f1_score_result*100))
print('Recall: %.2f%%' % (recall*100))
print("=====================")


model_save_name = 'classifier-XGBOOST.pt'
path = F"/content/drive/MyDrive/SP/{model_save_name}"
torch.save(model, path)

import discord
from discord.ext import commands,tasks
import os
from dotenv import load_dotenv
import torch

from sklearn import metrics
from sklearn import preprocessing
from sklearn.metrics import confusion_matrix,accuracy_score
from sklearn.linear_model import LogisticRegression

from tqdm import tqdm
import nltk
import re
import pickle
import torch

from nltk.stem.porter import PorterStemmer
from nltk.corpus import stopwords

from sklearn.feature_extraction.text import HashingVectorizer

load_dotenv()
TOKEN = os.getenv('TOKEN')
PATH = 'C:\\Users\\Ronell\\Desktop\\mafubot\\classifier.pt'

bot = commands.Bot(command_prefix='$', intents=discord.Intents.all())
bot.remove_command("help")

@bot.group(invoke_without_command= True)
async def help(ctx):
    embed= discord.Embed(title="Help", description="List of commands")
    embed.add_field(name="$assign", value="$assign [channel-name] (Accessible only by server owner). Assigns th
    embed.add_field(name="$info", value="Shows bot information and disclaimer")
    embed.add_field(name="$help", value="shows this page")
    await ctx.send(embed=embed)

def update_channel_id(channel):
    global CHANNEL_ID
    CHANNEL_ID = channel.id

@bot.event
async def on_message(message):

    message_iterable = []
    message_iterable.append(message.content)
    def preprocess_text(text):

        porter = PorterStemmer()

        def word_check(word):
            word_processed = porter.stem(word)
            word_processed="".join(letter for letter in word if letter.isalpha())
            return word_processed

        regrex_pattern = re.compile(pattern = "["
            u"\U0001F600-\U0001F64F"  # emoticons
            u"\U0001F300-\U0001F5FF"  # symbols & pictographs
            u"\U0001F680-\U0001F6FF"  # transport & map symbols
            u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                              "]+", flags = re.UNICODE)


        stop = stopwords.words('english')

        text = regrex_pattern.sub(r'',text)
        text = text.lower()
        return [word_check(word) for word in text.split() if word not in stop]

    vect = HashingVectorizer(decode_error='ignore', n_features=2**21,
                         preprocessor=None,tokenizer=preprocess_text,alternate_sign=False)

    X = vect.transform(message_iterable)
    model = torch.load(PATH)
    model_prediction = model.predict(X)
    model_probability_prediction = model.predict_proba(X)
    if message.author.bot == False:
        print('======================' + '\n\n')
        print('Message: ' + message.content)
        print('prediction value: ' + str(model_prediction))
        print(model_probability_prediction)
        print('Non-suicide probability prediction:  %.2f%%' % (model_probability_prediction[0][0]*100))
        print('Suicide probability prediction:  %.2f%%' % (model_probability_prediction[0][1]*100))
        print('\n\n' + '======================')
        # print(str(model_probability_prediction[0][1]) + " new")


    if model_prediction[0] == 1 and message.author.bot == False:

        channel = bot.get_channel(CHANNEL_ID)
        if channel is None:
            pass

        if channel is not None:
            await channel.send('Heads up! ' + str(message.author) + ' has been detected sending a message with 
                            'Message: ' + message.content + '\n'
                            'Probability: %.2f%%' % (model_probability_prediction[0][1]*100)
                            )

            print("==============================")


    if message.content == "hello":
        await message.channel.send("Hello!")



    await bot.process_commands(message)

@bot.command()
async def hello(ctx):
    await ctx.channel.send('Hello')

@bot.command()
async def assign(ctx, channel_name):
    if ctx.message.author == ctx.guild.owner:
        channel = discord.utils.get(ctx.guild.channels, name=channel_name)
        update_channel_id(channel)
        await ctx.channel.send('Channel ' + channel_name + ' successfully assigned!')
    else:
        await ctx.channel.send('Sorry, only the server owner has access to this action')

@bot.command()
async def info(ctx):
    channel = discord.utils.get(ctx.guild.channels)
    update_channel_id(channel)
    await ctx.channel.send('Welcome ' + str(ctx.message.author) + '!' + '\n\n' +
                           'This is SPBot! A machine learning powered suicidal text classification bot that det
                           'This bot uses a Logistic Regression model to evaluate messages being sent within th

                           'To start using the features of the bot, please start by using the $assign [channel-
                           'Note that only the server owner may use the $assign command.  It is done to safegua
                           'Using a private channel for admin roles is recommended for optimal usage \n\n\n' +

                           'DISCLAIMER: \n\n' +
                           'This model does not predict suicidal intent 100% of the time \n' +
                           'The model derived from testing only predicted suicidal messages 93.24% for all data
                           'This bot/tool is also not meant to replace actual diagnosis from domain experts but

                           'If you feel that you need help, please immediately contact a health professional fo

                           )


bot.run(TOKEN)
```

XI. Acknowledgment

First and foremost, I would like to thank my family who are always constantly providing for me. It is also because them that it was possible for me to have the privilege to study in this university. I also owe to them that I have the means to be able to work on this project and handle the stressful days while I'm working on it.

From the UP Manila faculty, thank you for opening up new concepts for me to learn and execute. They have taught me how to adapt and to constantly be open to learning new things. I would like to especially mention my past adviser, Mr. Berwin Yu, for helping me all throughout even though my first SP topic did not succeed, and for giving me advice not just for school, but for life as well. Of course, I would also like to show my gratitude and appreciation to my SP adviser, Doc Magboo, for taking me under your wing when I had decided to change my SP topic, and for guiding me with valuable insights and advice from day 1 of working on my topic, up to the day of the defense.

To my university friends! Siomai Rice, even though we got separated sa track, yung mga gala and gaming sessions natin were things that I was always looking forward to when we are still in univ. Kaya I would like to thank you for making univ life fun. To Enzo! We're almost always together during group projects, OJT, or classes man yan. Will always be grateful sa mga natutunan ko working with you (cringe), tara laro na!

To my JHS friends! Mga ka-Hatdog server, even though sometimes tilting yung games natin and may nagmamald (hindi ako yun!), I am genuinely thankful that you guys spare some time every night to play some fun games. Those sessions natin every night kinda served as my therapy every time nasstress ako from life, more games to come and sana lumago na ang mga twitch channels natin!

Lastly, I would like to thank myself. I always had a "bahala na" mentality on things, even sa defense ko. I have a habit of underestimating myself but the fact that I completed this I guess I can congratulate myself for a bit.