National College of Ireland

Analysis of Suicide Ideation Documents Posted on Twitter Using an NLP Classifier

MSc Research Project

Data Analytics-Group B

Rachana Swamy

ID: Student X21111413

School of Computing

National College of Ireland

Supervisor: Dr.Christian Horn

National College of Ireland

Configuration manual

Student Name: Rachana Devnur Swamy

Student ID: X21111413

Programme: Data Analytics **Year:** 2022

Module: Research project

Supervisor: Dr. Christian Horn

Submission Due Date: 15th December 2022

Project Title: Analysis of suicide ideation documents posted on Twitter using an NLP classifier

Word Count: 551

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Rachana Devnur Swamy

Date: 15-12-2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration manual

Analysis of suicide ideation documents posted on Twitter using an NLP classifier

Rachana Swamy

X21111413

1 Introduction

The following document will be covering instructions to reproduce the analysis of suicide ideation documents posted on Twitter using the Natural Language Processing Classifier and identify individuals with suicide ideation through their tweets.

2 System configuration

2.1 Hardware configuration

As for the hardware configuration, MacBook Air (M1 2020) has Mac OS Monterey, Version 12.4, with apple M1 chip and 8GB RAM, shown in figure 1.

Figure 1 Hardware configuration

2.2 Software Configuration

For the software configuration, majorly Jupiter notebook has been used. MS Excel has also been used to store the data. Figure two depicts a snapshot of Jupiter notebook which runs with help of Anaconda Navigator.

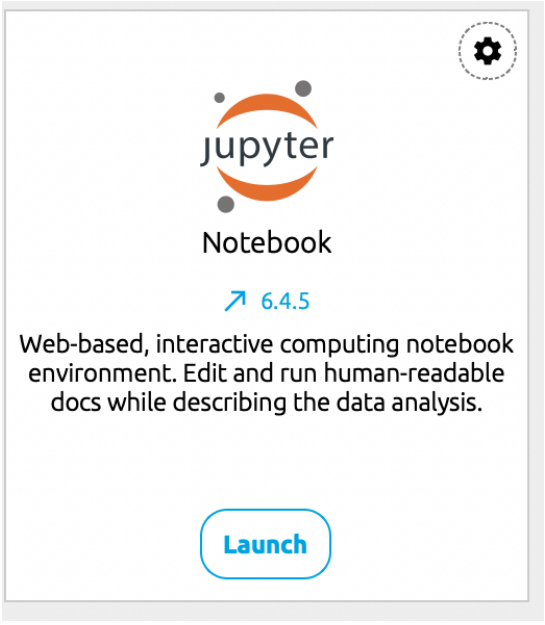

Figure 2 Software Configuration

3 Implementation

In order to increase the effectiveness of a classifier, raw data must be transformed into a more useful format. Since most tweets had high noise, the dataset was appropriately cleaned in this study before the job of detecting suicidal thoughts was carried out. The process of data pre-processing for textual analysis is shown in the below flow diagram

3.1 Data Source

1. Twitter Dataset: www.kaggle.com

2. The data now has to be pre-processed and cleaned as shown in snapshot 3

```
In [2]: def preprocess_tweet(text):
            text = re.sub('<[^>]*>', '', text)
            emoticons = re.findall('(?::|;|=)(?:-)?(?:\)|\(|D|P)', text)
            text = re.sub('[\W]+', ' ', text.lower())
            text = text+' '.join(emoticons).replace('-', '')
            return text
```

```
In [2]: tqdm.pandas()
        df = pd.read_csv('/Users/rachanaaradhya/Downloads/suicidal_data (1).csv')
        #df['tweet'] = df['tweet'].progress_apply(preprocess_tweet)
```

```
In [3]: df
```

Out[3]:

	label	tweet
0	1	my life is meaningless i just want to end my l...
1	1	muttering i wanna die to myself daily for a fe...
2	1	work slave i really feel like my only purpose ...

Figure 3 Code for Data cleaning and pre-processing

3. Since the tweets are long, to classify the text better, the below screenshot shows the tokenization of words

```
In [6]: from nltk.tokenize import RegexpTokenizer

        regexp = RegexpTokenizer('\w+')

        df['text_token']=df['tweet'].apply(regexp.tokenize)
        df.head(3)
```

Out[6]:

	label	tweet	text_token
0	1	my life is meaningless i just want to end my l...	[my, life, is, meaningless, i, just, want, to,...
1	1	muttering i wanna die to myself daily for a fe...	[muttering, i, wanna, die, to, myself, daily, ...
2	1	work slave i really feel like my only purpose ...	[work, slave, i, really, feel, like, my, only,...

Figure 4 Tokenization of words

4. After tokenization, a list of English stop words has been removed

```
In [7]: import nltk

        nltk.download('stopwords')
        from nltk.corpus import stopwords

        # Make a list of english stopwords
        stopwords = nltk.corpus.stopwords.words("english")
        df['text_token'] = df['text_token'].apply(lambda x: [item for item in x if item not in stopwords])
        df.head(3)

        [nltk_data] Downloading package stopwords to
        [nltk_data]     /Users/rachanaaradhya/nltk_data...
        [nltk_data]   Package stopwords is already up-to-date!
```

Out[7]:

	label	tweet	text_token
0	1	my life is meaningless i just want to end my l...	[life, meaningless, want, end, life, badly, li...
1	1	muttering i wanna die to myself daily for a fe...	[muttering, wanna, die, daily, months, feel, w...
2	1	work slave i really feel like my only purpose ...	[work, slave, really, feel, like, purpose, lif...

Figure 5 Removal of stop words

5. To still understand and detected the words better, words which have less than 2 alphabets have been taken off

```
In [8]: df['text_string'] = df['text_token'].apply(lambda x: ' '.join([item for item in x if len(item)>2]))

In [9]: df[['tweet', 'text_token', 'text_string']].head()
```

Out[9]:

	tweet	text_token	text_string
0	my life is meaningless i just want to end my l...	[life, meaningless, want, end, life, badly, li...	life meaningless want end life badly life comp...
1	muttering i wanna die to myself daily for a fe...	[muttering, wanna, die, daily, months, feel, w...	muttering wanna die daily months feel worthles...
2	work slave i really feel like my only purpose ...	[work, slave, really, feel, like, purpose, lif...	work slave really feel like purpose life make ...
3	i did something on the 2 of october i overdose...	[something, 2, october, overdosed, felt, alone...	something october overdosed felt alone horribl...
4	i feel like no one cares i just want to die ma...	[feel, like, one, cares, want, die, maybe, fee...	feel like one cares want die maybe feel less l...

Figure 6 Removal of stop words

6. Using frequency distribution, the words that have been repeated numerous times can be calculated.

```
In [11]: from nltk.probability import FreqDist

         fdist = FreqDist(tokenized_words)
         fdist

Out[11]: FreqDist({'want': 5010, 'dont': 4592, 'like': 3798, 'feel': 3476,
         47, 'ive': 2242, 'die': 2144, ...})
```

Figure 7 Frequently repeated words

7. Using the above dictionary, we can drop the words that are repeated less than 3-4 times

```
In [12]: df['text_string_fdist'] = df['text_token'].apply(lambda x: ' '.join([item for item in x if fdist[item] >= 1 ]))

In [13]: df[['tweet', 'text_token', 'text_string', 'text_string_fdist']].head()
```

Out[13]:

	tweet	text_token	text_string	text_string_fdist
0	my life is meaningless i just want to end my l...	[life, meaningless, want, end, life, badly, li...	life meaningless want end life badly life comp...	life meaningless want end life badly life comp...
1	muttering i wanna die to myself daily for a fe...	[muttering, wanna, die, daily, months, feel, w...	muttering wanna die daily months feel worthles...	muttering die daily months feel worthless shes...
2	work slave i really feel like my only purpose ...	[work, slave, really, feel, like, purpose, lif...	work slave really feel like purpose life make ...	work slave really feel like purpose life make ...
3	i did something on the 2 of october i overdose...	[something, 2, october, overdosed, felt, alone...	something october overdosed felt alone horribl...	something october overdosed felt alone horribl...
4	i feel like no one cares i just want to die ma...	[feel, like, one, cares, want, die, maybe, fee...	feel like one cares want die maybe feel less l...	feel like one cares want die maybe feel less l...

Figure 8 Dropped words which are repeated less than 2-4 times

8. With this, the words that could be repeated are sorted by lemmatization

```
In [15]: from nltk.stem import WordNetLemmatizer

         wordnet_lem = WordNetLemmatizer()

         df['text_string_lem'] = df['text_string_fdist'].apply(wordnet_lem.lemmatize)

In [16]: # check if the columns are equal
         df['is_equal']= (df['text_string_fdist']==df['text_string_lem'])
         df.is_equal.value_counts()

Out[16]: True     9114
         False       5
         Name: is_equal, dtype: int64
```

Figure 9 Lemmatization

9. A Word cloud is formed with the most repeated words

```
all_words_lem = ' '.join([word for word in df['text_string_lem']])
%matplotlib inline
import matplotlib.pyplot as plt
from wordcloud import WordCloud

wordcloud = WordCloud(width=600,
                      height=400,
                      random_state=2,
                      max_font_size=100).generate(all_words_lem)

plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off');
```

Figure 10 Word cloud formation

10. A plot is made to represent most repeated words

```
top_10 = fd.most_common(10)

# Create pandas series to make plotting easier
fdist = pd.Series(dict(top_10))
import seaborn as sns
sns.set_theme(style="ticks")

sns.barplot(y=fdist.index, x=fdist.values, color='blue');
```

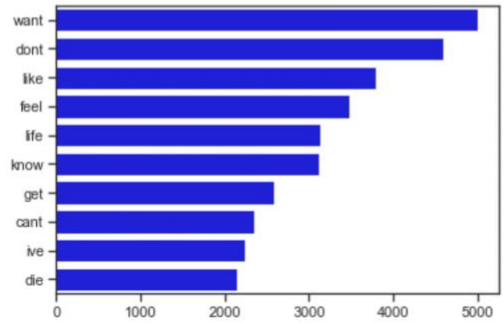

Figure 11 Graphical representation of the most repeated word

11. Sentiment analysis is done and is classified into Neutral, positive and negative tweets variables.

```
In [29]: from nltk.sentiment import SentimentIntensityAnalyzer

         analyzer = SentimentIntensityAnalyzer()

In [30]: df['polarity'] = df['text_string_lem'].apply(lambda x: analyzer.polarity_scores(x))
         df.tail(3)
```

Out[30]:

	label	tweet	text_token	text_string	text_string_fdist	text_string_lem	is_equal	polarity
9116	1	stop asking me to trust you when i m still cou...	[stop, asking, trust, still, coughing, water, ...	stop asking trust still coughing water last ti...	stop asking trust still coughing water last ti...	stop asking trust still coughing water last ti...	True	{'neg': 0.364, 'neu': 0.432, 'pos': 0.204, 'co...
9117	1	i never know how to handle sadness crying make...	[never, know, handle, sadness, crying, makes, ...	never know handle sadness crying makes feel st...	never know handle sadness crying makes feel st...	never know handle sadness crying makes feel st...	True	{'neg': 0.467, 'neu': 0.36, 'pos': 0.173, 'com...
9118	1	when cancer takes a life we blame cancer depre...	[cancer, takes, life, blame, cancer, depressio...	cancer takes life blame cancer depression dise...	cancer takes life blame cancer depression dise...	cancer takes life blame cancer depression dise...	True	{'neg': 0.891, 'neu': 0.109, 'pos': 0.0, 'comp...

```
In [33]: df = pd.concat(
             [df.drop([ 'polarity'], axis=1),
              df['polarity'].apply(pd.Series)], axis=1)
         df.head(3)
```

Out[33]:

	label	tweet	text_token	text_string	text_string_fdist	text_string_lem	is_equal	neg	neu	pos	compound
0	1	my life is meaningless i just want to end my l...	[life, meaningless, want, end, life, badly, li...	life meaningless want end life badly life comp...	life meaningless want end life badly life comp...	life meaningless want end life badly life comp...	True	0.357	0.467	0.177	-0.9154

Figure 12 Sentiment analysis

12. Tweets with the highest positive and highest negative statements are retrieved

```
In [37]: # Number of tweets
         sns.countplot(y='sentiment',
                       data=df,
                       palette=['#b2d8d8',"#008080", '#db3d13']
                       );
```

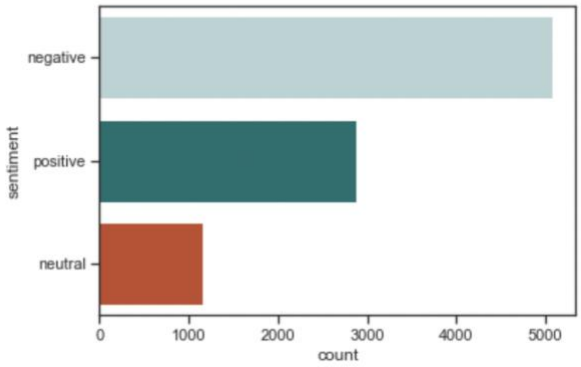

Figure 13 Negative, positive & neutral tweets

13. Word clouds for neutral positive and negative words are made

Figure 14 Example word cloud for neutral words (similarly made for negative & positive words)

14. Calculating the tweet's word length

```
: #Calculating tweet's lenght and word count
  df['text_len'] = df['text_string_lem'].astype(str).apply(len)
  df['text_word_count'] = df['text_string_lem'].apply(lambda x: len(str(x).split()))
  round(pd.DataFrame(df.groupby("sentiment").text_len.mean()),2)
```

:

sentiment	text_len
negative	298.46
neutral	32.40
positive	220.66

Figure 15 Tweet word count with respect to positive, negative & neutral words

15. N-grams, Bi-grams & Tri-grams are built to classify them better

```
('years old', 148),
('feel likei', 145),
('dont care', 143),
('hate want', 143),
('year old', 141),
('years ago', 139),
('want end', 138)]
```

```
#n3_trigram
n3_trigrams = get_top_n_gram(df['text_string_lem'

n3_trigrams
```

```
[('dont want die', 236),
 ('hate want die', 131),
 ('want die feel', 130),
 ('feel like consuming', 123),
 ('like consuming entire', 123),
 ('die feel blood', 121),
 ('feel blood boiling', 121),
```

Figure 16 Bi-grams & Tri-grams

3.2 Evaluation Methods

1. Data is prepared to train the models

```
In [62]: X = df["tweet"].to_list()
         y = df['label']
         from sklearn.model_selection import train_test_split
         X_train,X_test,y_train,y_test = train_test_split(X,
                                                          y,
                                                          test_size=0.20,
                                                          random_state=0)
         X_train = vect.transform(X_train)
         X_test = vect.transform(X_test)
         classes = np.array([0, 1])
```

```
In [65]: X_train.shape
Out[65]: (7295, 2097152)
```

Figure 17 Data preparation

2. For Machine learning classifiers, various performance measures have been represented

	precision	recall	f1-score	support
0	0.91	0.96	0.93	1060
1	0.94	0.87	0.90	764
accuracy			0.92	1824
macro avg	0.92	0.91	0.92	1824
weighted avg	0.92	0.92	0.92	1824

Figure 18 SVM Classification report

3. Logistic regression

	precision	recall	f1-score	support
0	0.89	0.95	0.92	1060
1	0.93	0.84	0.88	764
accuracy			0.91	1824
macro avg	0.91	0.90	0.90	1824
weighted avg	0.91	0.91	0.91	1824

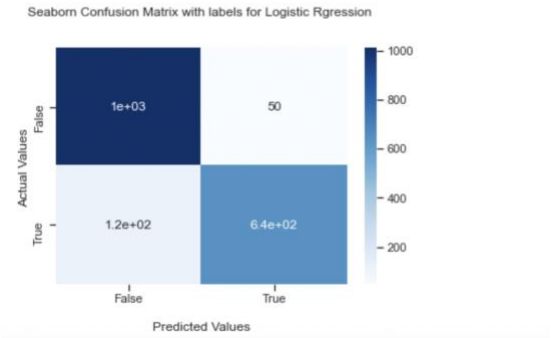

Figure 19 Logistic regression report and confusion matrix

4. SGD classifier

```
# SGD CLASSIFIER
from sklearn.linear_model import SGDClassifier
clf = SGDClassifier(loss='log', random_state=1)
classes = np.array([0, 1])
clf.partial_fit(X_train, y_train,classes=classes)
```

```
SGDClassifier(loss='log', random_state=1)
```

```
predictions_SGD = clf.predict(X_test)
print(classification_report(y_test, predictions_SGD))
```

	precision	recall	f1-score	support
0	0.89	0.96	0.93	1060
1	0.94	0.84	0.89	764
accuracy			0.91	1824
macro avg	0.92	0.90	0.91	1824
weighted avg	0.91	0.91	0.91	1824

Figure 20 SGD classification report

Testing and making predictions of the sentences on tweets

```
In [79]: # TESTING AND MAKINF PREDICTION ON THE SENTENCES
         label = {0:'negative', 1:'positive'}
         example = ["I'll kill myself am tired of living depressed and alone"]
         X = vect.transform(example)
         print('Prediction: %s\nProbability: %.2f%%'
               %(label[clf.predict(X)[0]],np.max(clf.predict_proba(X))*100))

         Prediction: positive
         Probability: 94.92%

In [80]: label = {0:'negative', 1:'positive'}
         example = ["It's such a hot day, I'd like to have ice cream and visit the park"]
         X = vect.transform(example)
         print('Prediction: %s\nProbability: %.2f%%'
               %(label[clf.predict(X)[0]],np.max(clf.predict_proba(X))*100))

         Prediction: negative
         Probability: 96.83%

In [81]: = {0:'negative', 1:'positive'}
         le = ["I am writing this kind of letter for the first time. My first time of a final
         ect.transform(example)
         ('Prediction: %s\nProbability: %.2f%%'
          %(label[clf.predict(X)[0]],np.max(clf.predict_proba(X))*100))

         Prediction: positive
         Probability: 79.95%
```

Figure 21 Predicting and testing tweets