Detection Of Suicidal Ideation From Social Media Comments Using Natural Language Processing Techniques

Group 19:
Pathyusha Jampala
Poojitha Pothini
Siva Manikanta Lakumarapu

Abstract

Our Social media is filled with different kinds of feed including entertainment, news, memes, pictures,sports or friends and stuff. Social media refers to being able to access and share information, ideas,feelings and news through virtual communication.Some of these thoughts be like feeling alone,feeling trapped, expressing how hopeless they are or feeling alone and so on.
So,Our aim is to develop a system for the early identification of suicide ideation based on his/her posts and comments on social media which helps to prevent suicide in the early stages.We use different machine learning and Natural Language Processing techniques to classify text into suicidal and non-suicidal text and detecting the risk of suicides.We train the model using the data to be able to classify the messages into suicidal or non-suicidal when given in the real-time.
We mainly design a web page to take a message like "I am happy" or "I want to leave this world" and so on from the user and predicts output whether the message is suicidal or non-suicidal.If the output is "suicidal",the page leads the user to a chat-bot to talk to and help the user to overcome such thoughts.

Design Overview:

Dataset:

We have taken a dataset "Suicide and Depression Detection" from kaggle and the link to the dataset is below:
https://www.kaggle.com/datasets/nikhileswarkomati/suicide-watch
The dataset is in the form of a CSV file which has two columns namely,
Text: It is the input and contains content from user post.It has 232074 unique values and data is the form of text.
Class:It is the target variable which has two unique values- suicide and non-suicide.

Importing Libraries:

Import pandas and numpy libraries for dataprocessing like loading the dataset which we are using to train the model.
Import Natural Language toolkit, PorterStemmer,pickle and string libraries for text processing.

Import different classifiers required to classify text into suicidal and non-suicidal from sklearn namely, randomForest, adaboost, GradientBoosting,Bagging,voting clssifier and Decision Tree classifier.Also import Naive_bayes, XGBoost classifiers for classifying the data and training the models.
Import Tfidf Vectorizer from sklearn for feature extraction.
Import select kBest,chi2,f_classif from sklearn for feature selection.
Import metrics from sklearn to analyze the performance and accuracy of the machine learning models.

Data Preprocessing:

In this step,load the dataset using pandas and perform data cleaning and preprocessing by removing null values.

Text Preprocessing:

The textual data preprocessing using Porterstemmer,nltk libaries may include removing accented characters (ex: cafe and café),expanding contractions(ex:can't and can not),removing whitespaces, converting to lowercases,fixing word length,removal of special characters and digits, spellings collection,removing stop words.Implements algorithms for extracting relevant features from the preprocessed data, crucial for subsequent analysis.
After Text preprocessing,save the cleaned dataset into a new file and load the file using pandas.

Before text preprocessing:

	text	class
74414	I Don't know??? Months self harm free and the ...	suicide
149516	I HAVE TO START BECOMING RICH I HAVE TO START ...	non-suicide
12484	A poem (haiku) for u/Me-Game-Dev hi, hello hel...	non-suicide
14043	I've honestly got no idea what to do anymore.I...	suicide
30673	Do you ever just cry? Like you just think abou...	non-suicide

After text preprocessing:

	text	class
74414	dont know7 month self harm free urg get strong...	suicide
149516	start becom rich start compani becom 16 afford...	non-suicide
12484	poem haiku umegamedev hi hello hello stop fuck...	non-suicide
14043	ive honestli got idea anymoreit feel everyon f...	suicide
30673	ever cri like think unfair life cri cant cri e...	non-suicide

Apply Tf-idf Vectorizer on the data to find out the frequency of each word in the dataset which generates a tf-idf matrix which can be used for training the machine learning models.we used TF-IDF vectorizer with a minimum document frequency of 50 and a maximum of 5000 features.
Divide the dataset in a ratio of 70:30 which 70% data of dataset is used to train the models while the remaining 30% is used to test the model.

Machine learning models:

Voting Classifier: We used the Combination of Gaussian, Bernoulli, and Multinomial Naive Bayes for training and testing the model with the preprocessed dataset.
Confusion matrix and classification report:

	precision	recall	f1-score	support
non-suicide	0.88	0.88	0.88	1542
suicide	0.87	0.87	0.87	1458
accuracy			0.88	3000
macro avg	0.88	0.88	0.88	3000
weighted avg	0.88	0.88	0.88	3000

Gradient Boosting: Utilization of a randomized search for hyperparameter optimization for training and testing the model with the preprocessed dataset.
Confusion matrix and classification report:

	precision	recall	f1-score	support
non-suicide	0.70	0.90	0.79	1542
suicide	0.85	0.58	0.69	1458
accuracy			0.75	3000
macro avg	0.77	0.74	0.74	3000
weighted avg	0.77	0.75	0.74	3000

XGBoost: Configuration with specific parameters for optimal performance.
Confusion matrix and classification report:

	precision	recall	f1-score	support
0	0.77	0.83	0.80	1542
1	0.81	0.74	0.77	1458
accuracy			0.79	3000
macro avg	0.79	0.79	0.79	3000
weighted avg	0.79	0.79	0.79	3000

Logistic regession: Simple logistic regression for comparison.
Confusion matrix and classification report:

	precision	recall	f1-score	support
non-suicide	0.89	0.93	0.91	1542
suicide	0.92	0.88	0.90	1458
accuracy			0.91	3000
macro avg	0.91	0.90	0.91	3000
weighted avg	0.91	0.91	0.91	3000

Convolutional Neural Network (CNN):

For CNN we import sequential models in keras from tensorflow.We also import Embedding,conv1d, GlobalMaxPooling1D, Dense layers from keras,Tokenizer and pad_sequences for text preprocessing.
Apply tokenizer o text column of our cleaned dataset and convert the text into sequence of integers which is machine understandable.Convert those sequence of integers into same length by adding pad_sequence.Split the data into 80% and 20%.Use the 20% of data for testing and 80% for training.We generate x_train and y_train for taining and x_test,y_test for testing and evaluation of model performance.
Now,for CNN model construction,first embed the encoded words into model,activate the 1D convolutional layer using activation='relu' function.then apply GlobalMaxPooling1D to reduce the data to single vector,After that activate sigmoid function.
After being done with the data we compile the model with binary cross-entropy loss and Adam optimizer.Fit the model by dividing data into five epochs with a batch size of 64 as data is quite large.
Confusion matrix and classification report:

```
Epoch 1/5
113/113 [==============================] - 157s 1s/step - loss: 0.4807 - accuracy: 0.7854 - val_loss: 0.2877 - val_accuracy: 0.8850
Epoch 2/5
113/113 [==============================] - 156s 1s/step - loss: 0.2249 - accuracy: 0.9147 - val_loss: 0.2243 - val_accuracy: 0.9150
Epoch 3/5
113/113 [==============================] - 153s 1s/step - loss: 0.1239 - accuracy: 0.9614 - val_loss: 0.2113 - val_accuracy: 0.9137
Epoch 4/5
113/113 [==============================] - 166s 1s/step - loss: 0.0672 - accuracy: 0.9828 - val_loss: 0.2141 - val_accuracy: 0.9075
Epoch 5/5
113/113 [==============================] - 171s 2s/step - loss: 0.0353 - accuracy: 0.9929 - val_loss: 0.2288 - val_accuracy: 0.9087
```

Save the best model:

Now save the best model into a pickle file as best_model.pkl.We saved Logistic regression and dumped it in the pickle file as pickle.dump (logistic_regression,f) for prediction in the app.
Define a preprocess function for the input taken from the user and apply text preprocessing techniques like converting to lower cases,removing punctuations,stopwords,stemming and transforming into vector form.

Define an app function which takes input from the user and apply preprocess function on the text, apply the votingclssifiers on the preprocessed text from the user and gives the ouput as whether the text is suicidal or non suicidal.

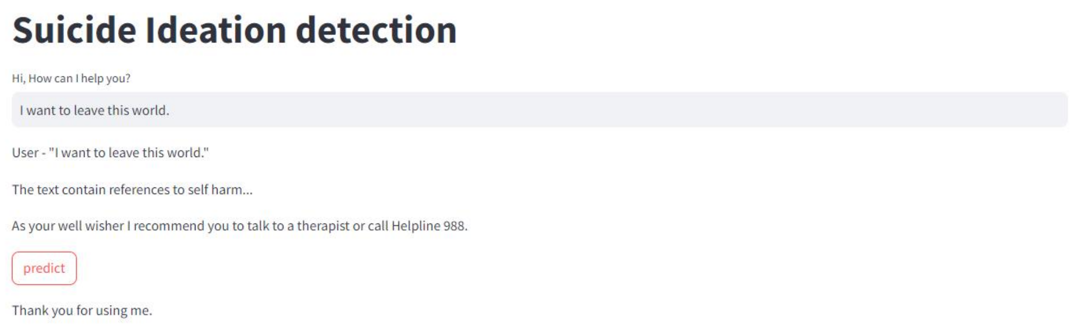
```
[ ] app('I am fetched up with my life i do not want to live anymore')

    Input :  I am fetched up with my life i do not want to live anymore
    Output :  suicide

[ ] app('poem haiku umegamedev hi hello hello stop fuck.')

    Input :  poem haiku umegamedev hi hello hello stop fuck.
    Output :  non-suicide
```

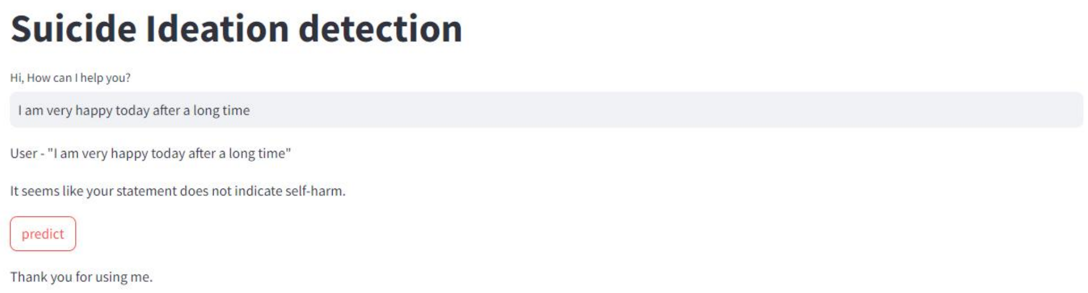
App.py

The app.py file to develop the web page for the user to enter his message for prediction.
First import streamlit library for creating web application, sklearn for data analysis,pickle for using the dumped model previously.
Define a preprocess() function to remove stop words,convert the alphabet to lowercase,and so on.Convert the data into array of integers and store it as inputToModel which is inputed to the model for prediction.
And open the best_model.pkl pretained model which was previously dumped in pickle.
Define a detection() function which takes user inputed text,apply preprocess() method for user entered text.After preprocessing the data is transformed into tf-idf format,,apply predict() function to classify the text as “suicidal” or “non-suicidal”.
Include the function we need to add in the web page with the title as “Mental Health app”.Define the input text box from which user can enter his message for pediction,using write() function of streamlit library and
If the model classifies the text as suicidal” it gives “The text cointain references to self-harm”,recommends a contact number to consult with and if it is “non-suicidal” it gives “The text does not contain self-harm”.
Use `!streamlit run app.py & npx localtunnel --port 8501`
Command to run the streamlit app which generates a link to open the web page.It generates a different link every time we run the command by deactivating the previous link.We can open the link on any system with the External URL link of our system which is 104.196.26.85.This is to protect from phishing the data from our system.

Final output:

Web application as below:





Anticipated Outcomes:

- We aim to develop a robust suicide detection system which can detect the text as suicidal or non-suicidal data.
- We intended to evaluate different machine learning models for text classification.
- We aim to design a model which predicts better outcome for suicidal thoughts detection which can help in reducing the suicides.

Actual Results:

- We developed a successful suicide detection web application with a maximum possible accuracy of 92%.
- Comparison of machine learning models, with the VotingClassifier,Logistic regression are showing promising results.
- Implementation and training of a Convolutional Neural Network for text classification.
- Evaluation of five different machine learning algorithms using their evaluation metrics such as accuracy,precision,recall and F1-score.

Result Analysis:

The Logistic Regression demonstrated the best overall performance with high precision, recall, and accuracy. It outperformed other models in detecting suicidal tendencies in textual data. The Convolutional Neural

Network also provided competitive results, showcasing its potential for text classification tasks.

Result evaluation:

Model	Precision	Recall	Accuracy	F1-score
Naive Bayes	Non-suicide:0.88 Suicide:0.87	Non-suicide:0.88 Suicide:0.87	0.88	0.88
Gradient Boosting	Non-suicide:0.79 Suicide:0.70	Non-suicide:0.67 Suicide:0.81	0.74	0.74
XG Boost	Non-suicide:0.77 Suicide:0.81	Non-suicide:0.83 Suicide:0.74	0.79	0.79
Logistic Regression	Non-suicide:0.89 Suicide:0.92	Non-suicide:0.93 Suicide:0.88	0.91	0.91
CNN	-	-	0.92	-

Project Code:

```
import warnings
warnings.filterwarnings('ignore')
#libries for importing and performing operations on data
import numpy as np
import pandas as pd
# Machine Learning Libraries
from sklearn.model_selection import train_test_split,GridSearchCV,RandomizedSearchCV
from sklearn.feature_selection import SelectKBest,chi2,f_classif
from sklearn.ensemble import
RandomForestClassifier,VotingClassifier,AdaBoostClassifier,GradientBoostingClassifier,BaggingClas
sifier
from sklearn.metrics import classification_report , confusion_matrix
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
import pickle
import string
#libraries for Text Processing
import nltk
nltk.download('stopwords')
nltk.download('punkt')
from nltk.stem import PorterStemmer
from google.colab import drive
drive.mount('/content/drive')
data = pd.read_csv('/content/Suicide_Detection.csv')
data.head()
data.shape
d_frame = data.sample(n=10000, random_state=42)
d_frame.info()
d_frame.drop(columns = 'Unnamed: 0',inplace=True)
d_frame.head()
d_frame['text']= d_frame['text'].str.lower()
d_frame['text'] = d_frame['text'].str.replace(r'[^\w\s]+', '',regex = True)
from nltk.corpus import stopwords
```

```
stop_words = stopwords.words('english')
d_frame['text'] = d_frame['text'].apply(lambda x: ' '.join([word for word in x.split() if word not in
(stop_words)]))
d_frame['text'] = d_frame['text'].apply(lambda x:nltk.word_tokenize(x))
ps = PorterStemmer()
d_frame['text'] = df['text'].apply(lambda x : [ps.stem(i) for i in x])
d_frame['text']=df['text'].apply(lambda x : ' '.join(x))
d_frame.head()
# Savecleaned dataset.
d_frame.to_csv('cleaned_data.csv')
newd_frame = pd.read_csv('cleaned_data.csv')
newd_frame.head()
ind = newd_frame[newd_frame['text'].isnull()].index
d_frame.iloc[ind]
newd_frame.dropna(inplace=True)
x,y = newd_frame['text'],newd_frame['class']
vectorizer = TfidfVectorizer(min_df=50,max_features=5000)
x =  vectorizer.fit_transform(x).toarray()
# Save the model
with open('tfidf.pkl', 'wb') as f:
   pickle.dump(vectorizer, f)
X_train,X_test,y_train,y_test = train_test_split(x,y,test_size=0.30,random_state=5)
X_train.shape,X_test.shape
nb = GaussianNB()
nb2 = BernoulliNB()
nb3 = MultinomialNB()
VotingClassifiers = VotingClassifier(estimators=[('GaussianNB', nb),('BernoulliNB',nb2),
('MultinomialNB', nb3)], voting = 'soft')
VotingClassifiers.fit(X_train, y_train)
print('Training score:',VotingClassifiers.score(X_train, y_train))
print('Testing score:',VotingClassifiers.score(X_test,y_test))
y_act=y_test
y_pred=VotingClassifiers.predict(X_test)
print(classification_report(y_act,y_pred))
model3 = RandomizedSearchCV(GradientBoostingClassifier(),{"learning_rate": range(3,5),
              "max_depth":[200],"max_features":range(6,10,2),
               "n_estimators":[10]},random_state=8,n_jobs=-1)
model3.fit(X_train,y_train)
print('Training score:',model3.score(X_train,y_train))
print('Testing score:',model3.score(X_test,y_test))
model3.best_params_
#confusion matrix and classification report
y_act=y_test
y_pred=model3.predict(X_test)
print(classification_report(y_act,y_pred))
model = XGBClassifier( eval_metric='map',max_depth=200,n_estimators=70,learning_rate=1.99)
model.fit(X_train,y_train.replace({"non-suicide":0,'suicide':1}))
print('Training score:',model.score(X_train,y_train.replace({"non-suicide":0,'suicide':1})))
print('Testing score:',model.score(X_test,y_test.replace({"non-suicide":0,'suicide':1})))
#matrix
y_act = y_test.replace({"non-suicide":0,'suicide':1})
y_pred = model.predict(X_test)
print(classification_report(y_act,y_pred))
from sklearn.linear_model import LogisticRegression
logistic_regression = LogisticRegression()
logistic_regression.fit(X_train, y_train)
y_pred = logistic_regression.predict(X_test)
print(classification_report(y_test, y_pred))
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
```

```
y_train_encoded = label_encoder.fit_transform(y_train)
print(set(y_train_encoded))
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
tokenizer = Tokenizer()
tokenizer.fit_on_texts(dfnew['text'])
X_sequence = tokenizer.texts_to_sequences(dfnew['text'])
X_padded = pad_sequences(X_sequence)            # Padding sequences to make of same length
X_train, X_test, y_train, y_test = train_test_split(X_padded, y, test_size=0.2, random_state=42)
# Creating Model
model = Sequential()
model.add(Embedding(input_dim=len(tokenizer.word_index)           +           1,           output_dim=128,
input_length=X_padded.shape[1]))
model.add(Conv1D(64, 3, activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(1, activation='sigmoid'))
# Compiling model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# Fitting of the model
model.fit(X_train, y_train_encoded, epochs=5, batch_size=64, validation_split=0.1)
# save the Model
with open('best_model.pkl', 'wb') as f:
   pickle.dump(logistic_regression, f)
def preprocess(inp):
   inp = inp.lower() #convert to lower case
   inp = inp.replace(r'[^\w\s]+', '') #remove punctuations
   inp = [word for word in inp.split() if word not in (stop_words)] #tokenize the sentence
   inp = ' '.join([ps.stem(i) for i in inp]) #stremming
   inputToModel = vectorizer.transform([inp]).toarray() #transform to vector form
   return inputToModel
def app(input_text):
   # Define the input text box
   print('Input : ',input_text) #take input from user
   processed_array = preprocess(input_text) #preprocess the text
   predict = VotingClassifiers.predict(processed_array) #Model prediction
   print('Output : ', predict[0])
%%writefile app.py
import streamlit as st
import sklearn
import pickle
import sklearn
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
stop_words = stopwords.words('english')
# better file handling needed
with open('tfidf.pkl', 'rb') as f:
   tfidf = pickle.load(f)
def preprocess(inp):
   inp = inp.lower()
   inp = inp.replace(r'[^\w\s]+', '')
   inp = [word for word in inp.split() if word not in (stop_words)]
   ps = PorterStemmer()
   inp = ' '.join([ps.stem(i) for i in inp])
   inputToModel = tfidf.transform([inp]).toarray()
   return inputToModel
# Load the pre-trained model
with open('best_model.pkl', 'rb') as f:
```

```
    model = pickle.load(f)
print('Done loading')
def detection(input_text):
    processed_array = preprocess(input_text)
    prediction = model.predict(processed_array)
    if prediction[0] == 'suicide':
        st.write("The text contain references to self harm...\n")
        st.write("As your well wisher I recommend you to talk to a therapist or call Helpline 988."
    elif prediction[0] == 'non-suicide':
        st.write(" It seems like your statement does not indicate self-harm.")
    else:
        st.write(" I couldn't make a clear prediction. Please provide more information.")
# Set the app title and heading
st.set_page_config(page_title='Suicide-Detection App', layout='wide')
st.title('Suicide Ideation detection')
# Define the input text box
input_text = st.text_input(' Hi, How can I help you?')
# Check if the user has entered a statement
if input_text:
    st.write(f"User - \"{input_text}\"")
    detection(input_text)
# Define the predict button
if st.button('predict'):
    st.write("Thank you for using me.")
!streamlit run app.py & npx localtunnel --port 8501
```

References:

[1]https://www.kaggle.com/code/rutujapotdar/suicide-text-classification-nlp
[2]https://github.com/gohjiayi/suicidal-text-detection
[3]https://www.linkedin.com/pulse/suicide-ideation-nlp-analysis-sergey-sundukovskiy