DS 5220–Supervised Machine Learning And Learning Theory

Project Report

Speech Detection using Twitter Data

By

002819131(Team 33)- Abhiram Varanasi

Submitted to

Professor Ryan Zhang

December 2024

Acknowledgement:

I would like to express my sincere thanks and gratitude to our Supervised Machine Learning and Learning Theory Professor Ryan Zhang for giving me the opportunity to work on this project. I'm very grateful for the support and guidance in completing this project.

NUID	***Student Name***
002819131	Abhiram Varanasi

Abstract:

Hate Speech classification has crucial applications in the social media domain. We describe the performance of our classifiers in the Hate Speech and Offensive Content Identification Track of FIRE 2021 conference.
The initial research in hate speech dates back to 1993 and has been legally defined by JohnT. Nockleby to describe any communication that incites hatred against anyone or any group of people in the name of race, ethnicity, religion, sexual orientation etc [1] Hate speech also poses challenges concerning the language used and the context in which it originates [2]. These challenges have made the hate speech identification task an interesting topic to be studied.

1. Introduction

Hate speech covers many forms of expressions which advocate, incite, promote or justify hatred, violence and discrimination against a person or group of persons for a variety of reasons. If left unaddressed, it can lead to acts of violence and conflict on a wider scale. In this sense hate speech is an extreme form of intolerance which contributes to hate crime.

Due to the increasing scale of social media, people are using social media platforms to post their views. It might be challenging to express harsh or disrespectful opinions to someone face-to-face. People therefore believe it is safe to abuse others or post offensive content online. As a result, they feel comfortable posting such material online. As a result, hate speech on social media is becoming more prevalent every day. In order to manage such a big number of users on social media, methods for automatic detection of hate speech are needed. In this study, we classify whether or not there is hate speech using machine learning techniques.

1.1 Objective and goal of the project

The goal of the project is to create a classification model which will classify tweets as hate speech or not. The main objective is to classify hate speech or not. We further will classify the hate speech tweet as hate, profane, offensive.

1.2 Problem Statement

Categorizing a phonocardiogram into either of two two states - normal or abnormal based on the present of irregular vibrations called heart murmurs.

2. Literature Survey

Pradeep et.al[6] explores the deep convolutional neural networks for the hate speech classification,the proposed Deep CNN model used the tweets with GloVe embedding method to semantic meaning with the help of convolution operation and achieved the

precision, recall and F1-score value as 0.97, 0.88, 0.92 respectively. The GloVe embedding is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and it helps to capture substructures within the word vector space model generated.

Watanabe et.al[7] proposed the idea of using semantic analysis to first classify the tweets as positive or negative,on the basis of the assumption that hate speech classification task is simplified as tweets containing hate speech are more likely to appear in a tweet classified as negative.SentiStrength software has been used to extract semantic-based features, semantic features, unigram features and pattern features,pattern features are extracted because they are useful to detect longer hateful expression. Therefore extracting patterns referring to words, as well as POS tags, ensures that one does not obtain exclusive patterns that apply to only very specific situations, but general ones that reflect hate regardless of the content.The classifiers used are random forest,SVM and J.48 and they give an accuracy equal to 87.4% on detecting whether a tweet is offensive or not (binary classification), and an accuracy equal to 78.4% on detecting whether a tweet is hateful, offensive, or clean (ternary classification).

Zhang et.al[9] proposed a new method based on a deep neural network combining convolutional and gated recurrent networks and implemented it large collection of publicly available twitter and other social media datasets, the model yielded competing results; a macro F1 score of 0.85 across those datasets, the author also pointed out that for classes (hate /not hate) having higher feature overlaps introduce bias which makes them easier to classify compared to the ones which have little to no overlapping.

Naseem et.al [8] proposed a Deep Context-Aware Embedding model for hate speech detection. This model has two modules: a deep hybrid contextual word representation and BiLSTM classifier. Deep hybrid contextual word representation is used for getting the word representation by concatenation of word and character level representations with context and lexicon level representation. This word representation is fed into the BiLSTM model for prediction. Authors used three datasets, dataset 1 contains total tweets 15,844 in which 1924 are labelled racism 3058 labelled sexism and remaining none . Dataset 2 contains 2512 tweets of which 1498 are labelled hateful, 19,326 are offensive and remaining neither. Dataset 3 contains 20,362 tweets of which 15,127 are labelled harassment and remaining none. This Deep Context-Aware Embedding model achieved an f1-score of 85.5, 92.3, 73.6 on dataset 1, dataset 2 and dataset 3 respectively.

A multi-task learning approach [10] has been proposed which involves in leveraging the shared affective knowledge to detect hate speech in Spanish tweets, using a well-known Transformer-based model, fundamentally it is a multilingual BERT model which is trained on Spanish texts;dubbed

BETO[11] In particular, this model integrates the Dynamic Masking technique in training which uses different masks for the same sentence for the data. For example, if the DM used is 10x, it means that every sentence had 10 different masks.
This model achieved macro precision score of 78.97, macro recall score 79.84 of and macro f1 score of 78.47.

Kent et.al[12] trained and tested a logistic regression classifier with 10 fold cross validation using n grams. GermEval 2018 Train dataset is used to train and development dataset for testing the model, train dataset contains 33.6% offensive tweets and development data contains 33.8% offensive tweets. They used Macro f1 as evaluation metric on character 1-2 , 1-3, 1-4, 1-5, 1-6 grams and achieved 65.98, 74.46, 75.49, 76.26, 39.83 for development dataset respectively.

3 Requirements Specification

3.1 Software Requirements

- Google Colab/ Jupyter Notebook to run the python modules
- Python 3
- Latest Python Libraries
- Training was done using the Tensorflow and keras frameworks

3.1 Hardware Requirements

Google Colab Compute Engine was used to train the model. It has the following specifications:

- Nvidia K80 or Nvidia P100 GPU depending on the allocation provided
- 2 vCPUs
- 24 GB of RAM
- Upto 16 GB of Graphics RAM

4 Methodology

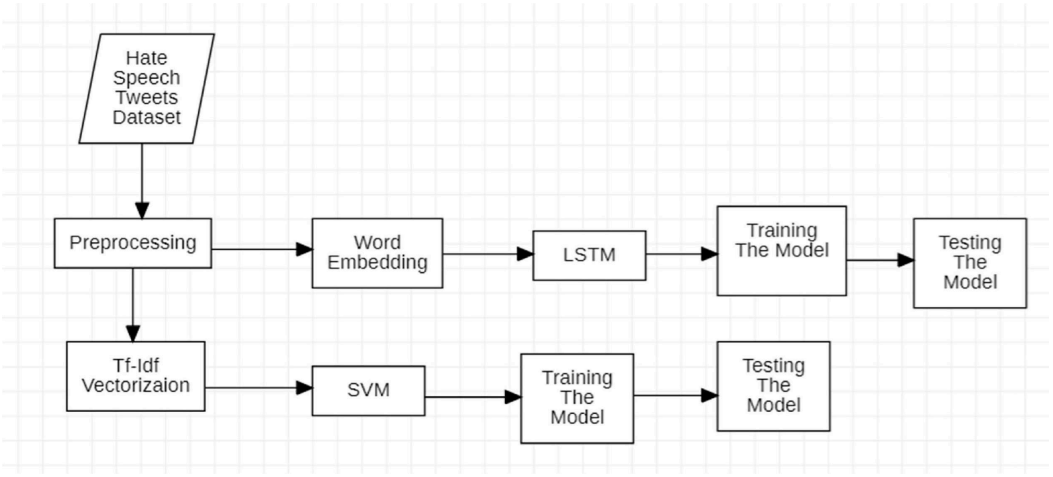


Figure 1 – Framework for binary classification (two models)

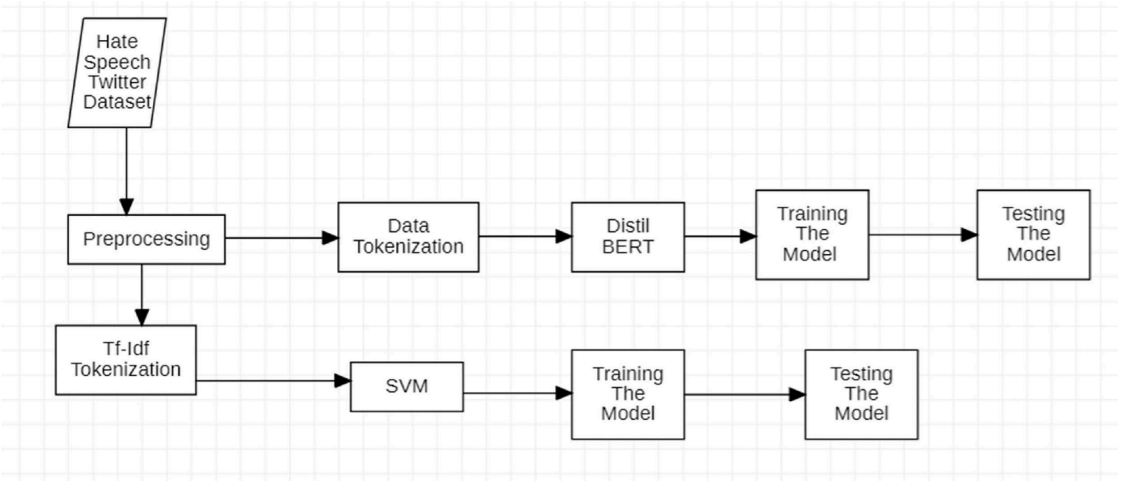


Figure 2 – Framework for multiclass classification (two models)

4.1.1 Data Analysis

Hasoc FIRE conference data consists of various hate tweets, this dataset seemed like a good problem because the nature of english tweets in this data is different from conventional english tweet data that are available online. Analysed two different tasks. The first task is a binary classification problem, which requires us to identify if a particular tweet is hateful or not. The second task is a complex one which requires

identifying if a tweet belongs to “Hate” or “Offensive” or “Profane” or “None” which is a multiclass classification problem.
The size of the dataset including training and test data is limited to about 4500 tweets.

4.1.2 Overcoming data shortage for neural networks

To overcome the data limitation we collected additional data from other sources. A free and publicly available Twitter hate speech dataset from Kaggle was chosen for data augmentation. This combined set is intended to be used for neural network models.
This Kaggle dataset has approximately thirty two thousand tweets with labels comparable to HOF and NOT. The HASOC training dataset contains 65% HOF tweets whereas the Kaggle dataset consists of only 45% HOF tweets .. The combined dataset has 40% of HASOC data and 60% of Kaggle data.
For the 2nd task The dataset provided by HASOC for multi-class classification of English tweets has 18% hate, 16% offensive, and 31% profane tweets.

4.2 Text Preprocessing for Hate Speech Tweets

We extensively analysed the tweets from the dataset to decide on the pre-processing step.Tweets have been either removed or have been transformed using pattern matching techniques to deem them fit to the classification models under consideration.

We have filtered out non-informative features from the tweets like URLs, white spaces,usernames that start with@ and hashtags. Other features like emojis have been filtered out.The tweets have been decontracted. Words like won’t,don’t,can’t,he’ll , I’ll etc have been converted to their complete forms. Stop words are those that appear very frequently in thetweets but don’t help in conveying any meaning. Common stop words like the, not, is and was have been removed. In addition, we have performed tokenization and lemmatization of the preprocessed tweets.The word cloud above denotes the important words within the hasoc data after preprocessing.

4.3 TF-IDF

TF-IDF vectorization[3] is a vector space model which converts tokens(words) into word vectors based on term frequency and inverse document frequency.
Term frequency:
1.tokenize the text, find the tokens, now generate term frequency vectors for each document.

$$TF(w,d) = \frac{occurences\ of\ w\ in\ document\ d}{total\ number\ of\ words\ in\ document\ d}$$

Inverse document frequency vector for each document:

$$IDF(w,D) = \ln(\frac{Total\ number\ of\ documents\ (N)\ in\ corpus\ D}{number\ of\ documents\ containing\ w})$$

NOW FIND THE DOT PRODUCT OF THE TF AND IDF MATRICES. We have vectorized to extract N GRAMS.(here we extracted unigrams)

4.4 Model selection

For Binary classification

We aim to work with simple machine learning models like SVM[4]+Tfidf+Character+NGRAM for binary classification since character n grams have an advantage in these tasks. We also explored how Recurrent neural networks like Long short term memory[5] can perform on this task by using word embeddings.For fine-grained multiclass classification task on the hasoc data, we aim to use Bidirectional Encoder representation using Transformer (BERT) model. Fine-tuned version of the BERT model has already been proven to perform well at classic NLP tasks like sentiment analysis. For binary classification we will be using HASOC+KAGGLE data since the standalone hasoc data is very limited in size and cannot be used for LSTM'S. For multi-class classifications we would like to use the

standard hasoc data without the kaggle data, as multiclass labelled data is unavailable with the same kind of labels.

For fine-grained multiclass classification

Turned to DistilBERT [13]. Google provides pre-trained partial BERT [14] language models which may be fine tuned based on our application. BERT is designed to pre-train deep bidirectional representations from the unlabeled text by jointly conditioning on both left and right context words. BERT uses the advantages of transfer learning, a method where a model developed for a task is reused as the starting point for a model on a second task. I have built DistilBERT, a model trained in a self-supervised fashion using the BERT base model as a teacher. It is smaller and faster than BERT. We have trained DistilBERT on the hate speech corpus [15]. The main reason for choosing this model, unlike other sequence classifiers like RNN, is that the DistilBERT model works on the entire sequence at once instead of reading the tokens sequentially. This process can be further accelerated usingGPU support provided by Google Colab. We could adapt the pre-trained DistilBERT model bytraining it further on our relatively smaller dataset, further fine-tuning the parameters for better accuracy without much computational overhead. The tweets in the dataset were of different lengths. We used padding to normalize all the tweets to have the maximum sequence length.TheDistilBERT model we chose was pre-trained on unlabelled Wikipedia corpus. It was then fine-tuned for our corpus by adding the output layer.TheDistilBERT [15] model we chose facilitated in developing a multi label regression classifier.The output vector from the model is a 4-dimensional numeric vector. We used one-hot encoding on the class labels of the existing HASOC dataset to obtain a 4-dimensional boolean output vector for each tweet. The first value of the vector indicates hate class, the second profane,the third offensive and the fourth represents neutral. For eg., a tweet that is profane will be encoded [0 1 0 0]. While classifying a tweet using DistilBERT, on obtaining the 4 dimensional output vector, the tweet is assigned the class corresponding to the vector position which has the highest numerical value.However, for the sake of comparison we tested SVM for multiclass classification as well and the results have been discussed.

5 Implementation

For binary classification:Initially only HASOC training data was used. The same data cleaning process as discussed in the previous sections has been used or all the data involved for both binary classification and multiclass classification. For Support vector machines, the text data has been vectorized using TF-IDF vectorization. The training data has been split into 80:20 for validation. The SVM model has been experimented with all types of kernels namely (linear,polynomial,sigmoid,radial basis function). The training data has been fit into the model and it has been tested on validation data. Out of which it was found that linear SVM has given the best performance. So linear SVM has been used to test on the HASOC testing data.The SVM has also been experimented on HASOC+KAGGLE data by following the same procedure as mentioned above.

For the LSTM network,TF-IDF vectorization has not been used, since the model accepts only word embeddings as input. The LSTM network has also been rained and tested using the same procedure mentioned for the SVM.

For multiclass classification: Since the multiclass labels are not available for The kaggle data, only HASOC data has been used for multiclass classification. The DistilBERT model has been trained on the HASOC training dataset and has been tested on the HASOC test data set. For the sake of experimentation even SVM and LSTM models have been used for this task, and we observed that the DistilBERT model outperformed them.

Metrics for binary classification:
Accuracy: The portion of the total sample which was predicted correctly is called accuracy.

$$Accuracy = \frac{TruePositive + TrueNegative}{TotalSample}$$

Precision: Precision is the ratio between the True Positives and all the Positives.

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

Recall:The recall is the measure of our model correctly identifying True Positives.

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

F1-score: F1-scores is the weighted average of recall and precision. It takes false positive and false negative into account while calculating.

$$\text{F1} = 2 \times \frac{Precision * Recall}{Precision + Recall}$$

Metrics for multiclass classification

For multiclass labels we use ***Macro-precision, Macro-recall and Macro-F1***.
For calculating the above metric we first calculate per-class label precision and recall using the above formula mentioned previously. Using precision and recall for each label we calculate F1-score for each label.Then

Macro precision = average of precision of all classes
Macro recall = average of recall of all classes Macro
F1-score = average of F1-score of all classes

Hyperparameters:

The following hyperparameters were used for training the models:

- SVM (Linear Kernel):
  - Kernel: Linear
  - Regularization parameter (C): 1.0
  - Maximum iterations: 1000
- LSTM:
  - Embedding dimension: 32
  - Sequence length: 130

    - Dropout: 0.2
    - Optimizer: Adam
    - Learning rate: 0.001
- DistilBERT:
    - Batch size: 16
    - Learning rate: 2e-5
    - Sequence length: 128
    - Epochs: 4

Dataset Insights:

The combined dataset comprised 40% HASOC data and 60% Kaggle data, creating a balanced distribution of classes. Key observations included:

- The most frequent hate-related words in the dataset included "racism," "hate," and "abuse."
- Offensive tweets often contained abbreviations or slang, requiring careful preprocessing.
- A word cloud of the processed tweets highlights prominent terms used in hateful and offensive language.

6. Results and Discussion

For binary classification:

Kernel	Macro F1	Macro precision	Macro Recall	Accuracy
Linear	0.73	0.76	0.72	0.79
RBF	0.41	0.34	0.50	0.69
Sigmoid	0.41	0.34	0.50	0.69
Polynomial	0.41	0.34	0.50	0.69

Table 1. This table describes performance of various kernels of SVM on HASOC only validation data

Kernel	Macro F1	Macro precision	Macro Recall	Accuracy
Linear	0.74	0.76	0.73	0.77
RBF	0.38	0.31	0.50	0.62
Sigmoid	0.38	0.31	0.50	0.62
Polynomial	0.38	0.31	0.50	0.62

Table 2. This table describes performance of various kernels of SVM on HASOC test data when only trained on HASOC data(refer table 1 for validation performance)

It is clearly evident from table 1 and table 2 that SVM classifier with Linear kernel outperformed the remaining 3 kernels by a significant margin,it is because The linear kernel[16] is good when there are a lot of features. That's because mapping the data to a higher dimensional space does not really improve the performance.In text classification, both the numbers of instances (document) and features (words) are large.

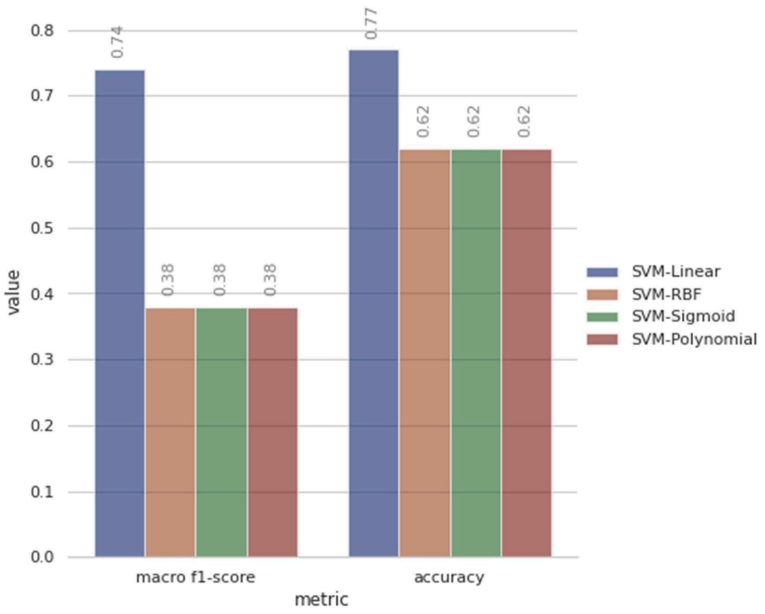
Kernel	Macro F1	Macro precision	Macro Recall	Accuracy
Linear	0.80	0.86	0.75	0.92
RBF	0.46	0.43	0.50	0.87
Sigmoid	0.46	0.43	0.50	0.87
Polynomial	0.46	0.43	0.50	0.87

Table 3. This table describes performance of various kernels of SVM on HASOC+KAGGLE validation data

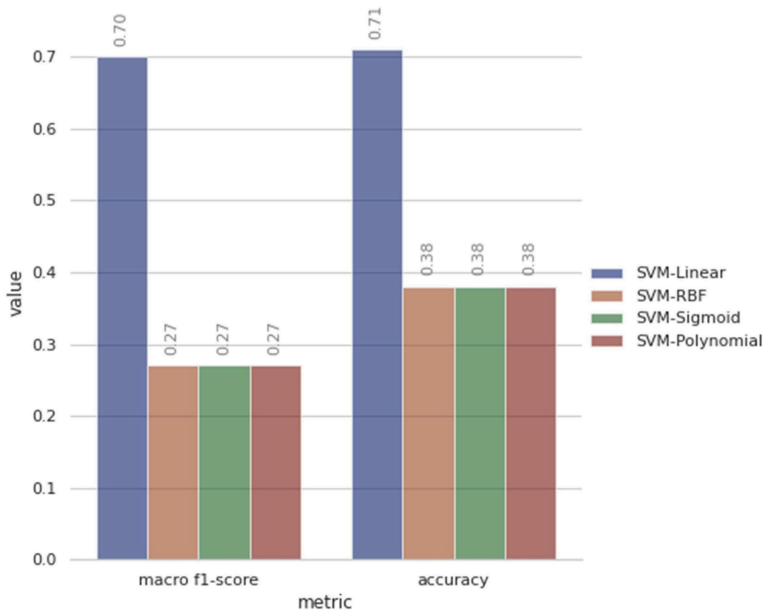
Kernel	Macro F1	Macro precision	Macro Recall	Accuracy
Linear	0.70	0.70	0.72	0.71
RBF	0.27	0.19	0.50	0.38
Sigmoid	0.27	0.19	0.50	0.38
Polynomial	0.27	0.19	0.50	0.38

Table 4. This table describes performance of various kernels of SVM on HASOC test data when trained on HASOC + KAGGLE(refer table 3 for validation performance)

From the above table we can infer that our decision to add more data to mitigate class imbalance in the previous HASOC only data has not given a positive response. The validation score shows that adding kaggle data to the initial hasoc data has dissipated the nature of the entire data as the size of kaggle data is comparatively higher compared to hasocdata.The testing metrics show a decrease in the best performing kernel.



Graph 1. This graph describes performance of various kernels of SVM on HASOC test data when only trained on HASOC data



Graph 2. This graph describes performance of various kernels of SVM on HASOC test data when trained on HASOC + KAGGLE

F1 Score	Accuracy
0.61	0.75

Table 5. This table describes the performance of LSTM network on validation data when only HASOC training data is used

F1 Score	Accuracy
0.61	0.71

Table 6. This table describes performance of LSTM network on HASOC test data when trained on HASOC only(refer table 5 for validation performance)

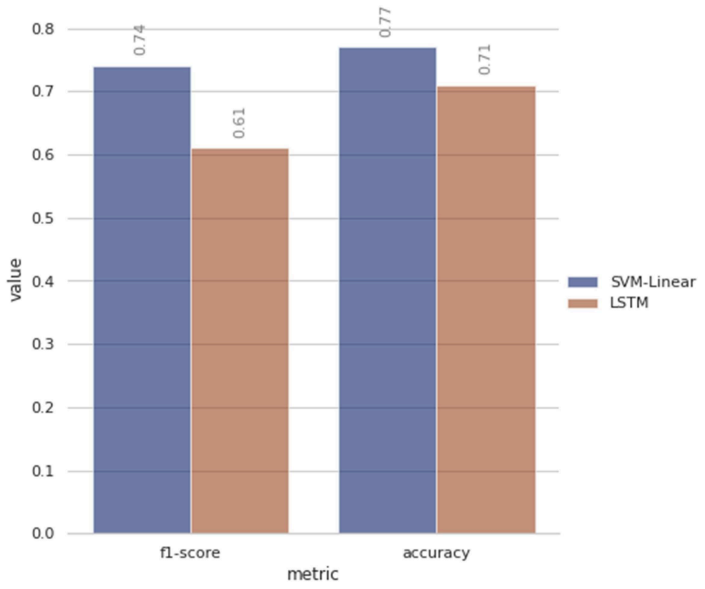
F1 Score	Accuracy
0.92	0.86

Table 7. This table describes the performance of LSTM network on validation data when only HASOC+KAGGLE training data is used

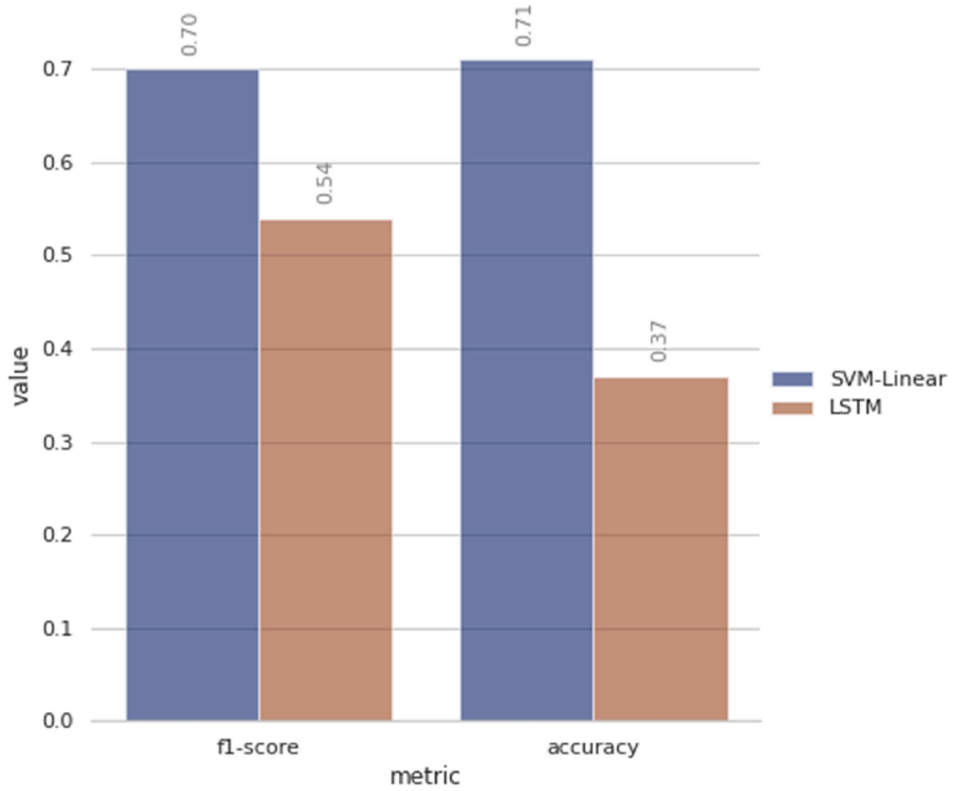
F1 Score	Accuracy
0.54	0.37

Table 8. This table describes performance of LSTM network on HASOC test data when trained on HASOC + KAGGLE(refer table 7 for validation performance)

Here we can observe that for binary classification, LSTM performed poorly compared to the Linear SVM(table 1,2). However, adding kaggle data and training LSTM on the combined data tells us that the nature of the data has a direct impact on the performance of LSTM.This makes it an important observation about data integration and model selection.



Graph 3. This graph describes performance of LSTM network and SVM-Linear on HASOC test data when trained on HASOC only



Graph 4.This graph describes performance of LSTM network and SVM-Linear on HASOC test data when trained on HASOC + KAGGLE

For multi class classification:

Kernel	Macro F1	Macro precision	Macro Recall	Accuracy
Linear	0.58	0.61	0.58	0.64
RBF	0.12	0.08	0.25	0.31
Sigmoid	0.12	0.08	0.25	0.31
Polynomial	0.12	0.08	0.25	0.31

Table 9. This table describes performance of various kernels of SVM on HASOC only validation data

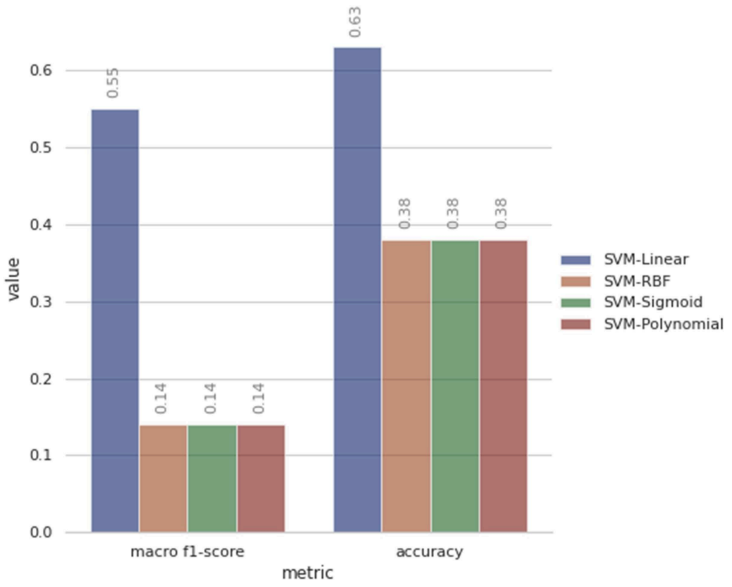
Kernel	Macro F1	Macro precision	Macro Recall	Accuracy
Linear	0.55	0.58	0.56	0.63
RBF	0.14	0.09	0.25	0.38
Sigmoid	0.14	0.09	0.25	0.38
Polynomial	0.14	0.09	0.25	0.38

Table 10. This table describes performance of various kernels of SVM on HASOC test data when only trained on HASOC data(refer table 1 for validation performance)

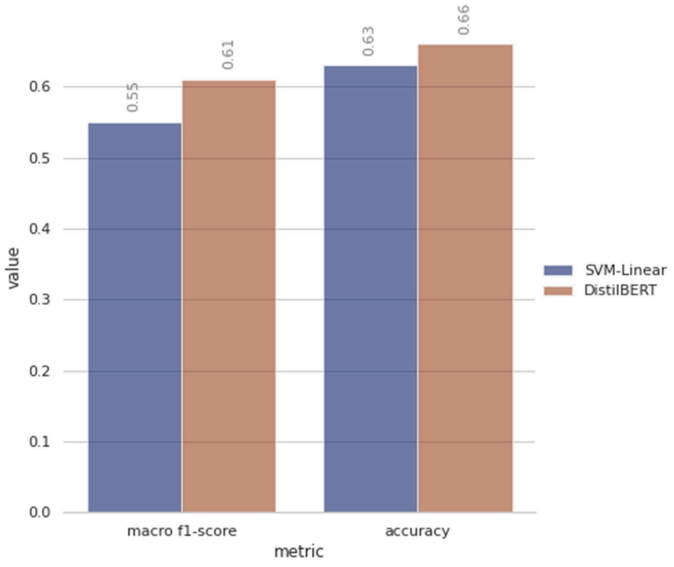
Macro F1	Macro precision	Macro Recall	Accuracy
0.61	0.61	0.61	0.66

Table 11. This table describes performance of DistilBERT model on HASOC test data when trained on HASOC data only.

For the multiclass classification, only SVM and DistilBERT models have shown better performance during our evaluation and testing process.It is clearly evident that DistilBERT outperformed SVM because it learned better representation of the text data.Though SVM could perform better compared to neural networks when class labels are less, it is noted that increase in the number of class labels showed a significant impact on the performance of the SVM.However, DistilBERT performed withstood and performed better than the Linear SVM.



Graph 5. This graph describes performance of various kernels of SVM on HASOC test data when only trained on HASOC data



Graph 6. This graph describes performance of DistilBERT model and SVM-Linear on HASOC test data when trained on HASOC data only.

7. Conclusion and Future Work

It is observed that care must be taken when it is decided that adding new data can solve the class imbalance problem. While oversampling techniques do exist and they preserve the nature of the data, using them to solve class imbalance might also not be

efficient as the dimensionality of the data should be kept in mind,as it has a greater impact on time and resource utilization for resampling techniques. The Linear SVM performs better when we are dealing with text data especially when we have a larger amount of features. It is also noted that the nature of the data has a great effect on the performance on LSTM as lstm performed better on hasoc only data compared to HASOC +KAGGLE data.Nature of the data implies, the way language is used in the data, it might differ depending from where the data has been collected from, what region are the people who generate the text come from etc.DistilBERT performed significantly better compared to Linear SVM for multiclass classification. In the future we would like to experiment more with these models and find opportunities where we could improve their performance.We would also like to use explainer tools like LIME to evaluate the models that we used.

8. REFERENCES

[1] J. Nockleyby, ‘hate speech in encyclopedia of the american constitution, Electronic Journalof Academic and Special librarianship (2000).

[2]Z. Zhang, L. Luo, Hate speech detection: A solved problem? the challenging case of longtail on twitter, Semantic Web 10 (2019) 925–945.

[3] J. Ramos, et al., Usingtf-idf to determine word relevance in document queries, in:Proceedings of the first instructional conference on machine learning, volume 242, Citeseer,2003, pp. 29–48.

[4] Abro, Sindhu, et al. "Automatic Hate Speech Detection using Machine Learning: A Comparative Study." Machine Learning 10.6 (2020).

[5] Staudemeyer, Ralf C., and Eric Rothstein Morris. "Understanding LSTM--a tutorial into Long Short-Term Memory Recurrent Neural Networks." arXiv preprint arXiv:1909.09586 (2019).

Unigram and SVM

In [ ]:

```
from google.colab import
drive
drive.mount('/content/drive
')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, c
all drive.mount("/content/drive", force_remount=True).
```

In [ ]:

```
import numpy as np
import pandas as
pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,f1_score, confusion_matrix
import re
from nltk.stem import WordNetLemmatizer
from keras.preprocessing.text import Tokenizer
import nltk
from        sklearn.feature_extraction.text        import
TfidfVectorizer    from    sklearn.feature_extraction.text
import CountVectorizer nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[ ]:

```
True
```

In [ ]:

```
from sklearn.preprocessing import LabelEncoder
```

In [ ]:

```
from sklearn.svm import SVC
```

TRAINING AND TEST SET

In [ ]:

```
hasoc_train=pd.read_csv("/content/drive/MyDrive/NLP
 PROJECT/en_Hasoc2021_train.csv") # hasoc only data
# combined data of hasoc+kaggle
hasoc_test=pd.read_csv("/content/drive/MyDrive/NLP
PROJECT/en_Hasoc2021_test_data.csv")
# hasoc test data
hasoc_test_bin=pd.read_csv("/content/drive/MyDrive/NLP
PROJECT/1A_English_actual_label s.csv") # hasoc binary test labels
hasoc_test_mul=pd.read_csv("/content/drive/MyDrive/NLP
PROJECT/1B_English_actual_label s.csv") #hasoc multiclass test labels
```

In [ ]:

```
combined_train=pd.read_csv("/content/drive/MyDrive/NLP
PROJECT/hate_bin_combine.csv",en coding = "ISO-8859-1")
```

In [ ]:

```
hasoc_train
```

Out[ ]:

	Unnamed: 0	_id	text	task_1	task_2
0	4986	60c5d6bf5659ea5e55defa2c	@wealth if you made it through this &...	HOF	PRFN
1	3394	60c5d6bf5659ea5e55def461	Technically that's still turning back the cloc...	HOF	OFFN
2	1310	60c5d6bf5659ea5e55defaad	@VMBJP @BJP4Bengal @BJP4India @narendramodi @J...	NOT	NONE
3	3390	60c5d6bf5659ea5e55def419	@krtoprak_yigit Soldier of Japan Who has dick ...	HOF	OFFN
4	4626	60c5d6bf5659ea5e55def7fa	@blueheartedly You'd be better off asking who ...	HOF	OFFN
...	...	...	...	...	...
3838	1661	60c5d6bf5659ea5e55defd57	@BBCNews Let the dog deal with the wanker once...	HOF	PRFN
3839	194	60c5d6bf5659ea5e55def185	India has suffered a lot. That Chinese bastard...	HOF	HATE
3840	3988	60c5d6bf5659ea5e55def78c	People didn't give 300+ seats majority to BJP ...	HOF	HATE
3841	4212	60c5d6bf5659ea5e55defb04	@KanganaTeam This is such a vile, xenophobic a...	HOF	PRFN
3842	1512	60c5d6bf5659ea5e55defb8a	@30iPpgStmILw0SI @ChinaDaily #ChineseVirus #Wu...	NOT	NONE

3843 rows × 5 columns

In [ ]:

```
combined_train
```

Out[ ]:

	_id	text	task_1
0	60c5d6bf5659ea5e55defa2c	@wealth if you made it through this &...	HOF
1	60c5d6bf5659ea5e55def461	Technically that's still turning back the cloc...	HOF
2	60c5d6bf5659ea5e55defaad	@VMBJP @BJP4Bengal @BJP4India @narendramodi @J...	NOT
3	60c5d6bf5659ea5e55def419	@krtoprak_yigit Soldier of Japan Who has dick ...	HOF
4	60c5d6bf5659ea5e55def7fa	@blueheartedly You'd be better off asking who ...	HOF
...	...	...	...
35800	31958	ate @user isz that youuu? Ã Â°Ã Â Ã Â Ã Â Ã Â Ã Â°Ã...	NOT
35801	31959	to see nina turner on the airwaves trying to w...	NOT
35802	31960	listening to sad songs on a monday morning otw...	NOT
35803	31961	@user #sikh #temple vandalised in in #calgary,...	HOF
35804	31962	thank you @user for you follow	NOT

35805 rows × 3 columns

binary test data

In [ ]:

```
hasoc_test["task_1"]=hasoc_test_bin["label"]
```

multiclass test data

In [ ]:

```
hasoc_test["task_2"]=hasoc_test_mul["label"]
```

In [ ]:

```
hasoc_test.head()
```

Out[ ]:

	_id	text	task_2	task_1
0	60c5d6bf5659ea5e55deffcb	Fewer people coming in for vaccinations. So sa...	NONE	NOT
1	60c5d6bf5659ea5e55df028c	@MattHancock This may all be true. But... What...	PRFN	HOF
2	60c5d6bf5659ea5e55def377	@Layla_EFC I've unfollowed him the wanker	PRFN	HOF
3	60c5d6bf5659ea5e55def4c7	You guys are losing it all over the world. The...	NONE	NOT
4	60c5d6bf5659ea5e55df01a6	And thus death laughs... It is sad merriment, ...	NONE	NOT

PREPROCESSING AND CLEANING

1. **LOWER CASE CONVERTING**
2. **REMOVING BAD SYMBOLS**
3. **STOP WORD REMOVER**
4. **PUNCTUATION REMOVER**
5. **TOKENIZATION**
6. **LEMMATIZATION**
7. **REMOVE USERNAMES**

In [ ]:

```
lemmatizer = WordNetLemmatizer()
stop_words= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselve
s', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 't
hey', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
at'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
d', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as'
, 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through'
, 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'ov
er', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
y', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too'
, 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
w', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'migh
tn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
asn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't","not"])
```

In [ ]:

```
def preprocess(tweets):

  tweets.encode('ascii', 'ignore').decode('ascii') #remove
  emojis tweets=tweets.lower() #convert to lower case tweets =
  re.sub(r"http\S+", " ", tweets) #remove urls
  l=tweets.split(" ")
  for n,i in enumerate(l):
    if '@' in i:
      l[n]=""

  tweets=" ".join(l)
  tweets= re.sub('[^A-Za-z]+',' ',tweets) #remove bad character
  tweets = [word for word in tweets.split(" ") if not word in stop_words] # removing
st op words
  tweets= [lemmatizer.lemmatize(token, "v") for token in tweets] #Lemmatization
  tweets=" ".join(tweets)
  return tweets
def decontract(text):
    text = re.sub(r"won\'t", "will not",
    text) text = re.sub(r"can\'t", "can
    not", text) text = re.sub(r"n\'t", "
    not", text)
    text = re.sub(r"\'re", " are", text)
    text = re.sub(r"\'s", " is", text)
    text = re.sub(r"\'d", " would", text)
    text = re.sub(r"\'ll", " will", text)
    text = re.sub(r"\'t", " not", text)
    text = re.sub(r"\'ve", " have", text)
    text = re.sub(r"\'m", " am", text)
    return text
```

PREPROCESSING THE TRAIN DATA

WHILE USING ONLY HASOC DATA

In [ ]:

```
hasoc_train["text"]=hasoc_train["text"].apply(lambda
x:decontract(x))
hasoc_train["text"]=hasoc_train["text"].apply(lambda
x:preprocess(x))
```

In [ ]:

```
train_df, val_df = train_test_split(hasoc_train, test_size = 0.2, random_state =
42) train_tweet=train_df.text
val_tweet=val_df.text
```

WHILE USING COMBINED DATA

In [ ]:

```
combined_train["text"]=combined_train["text"].apply(lambda
x:decontract(x))
combined_train["text"]=combined_train["text"].apply(lambda
x:preprocess(x))
```

In [ ]:

```
train_df, val_df = train_test_split(combined_train, test_size = 0.2, random_state =
42) train_tweet=train_df.text
val_tweet=val_df.text
```

PREPROCESSING THE TEST DATA

In [ ]:

```
hasoc_test["text"]=hasoc_test["text"].apply(lambda
x:decontract(x))
hasoc_test["text"]=hasoc_test["text"].apply(lambda
x:preprocess(x))
```

In [ ]:

```
test_tweet=hasoc_test.text
```

on evaluation data

In [ ]:

```
le=LabelEncoder()
```

for binary data

In [ ]:

```
train_df["task_1"] =
le.fit_transform(train_df["task_1"]) val_df["task_1"]
= le.fit_transform(val_df["task_1"])
```

for multiclass data

In [ ]:

```
train_df["task_2"] =
le.fit_transform(train_df["task_2"]) val_df["task_2"]
= le.fit_transform(val_df["task_2"])
```

In [ ]:

```
full_train_tweet=hasoc_train["text"]
```

In [ ]:

```
train_tweet=train_df["text"]
```

In [ ]:

```
val_tweet=val_df["text"]
```

on test data

for binary data

In [ ]:

```
hasoc_test["task_1"]=le.fit_transform(hasoc_test["task_1"])
```

for multiclass data

In [ ]:

```
hasoc_test["task_2"]=le.fit_transform(hasoc_test["task_2"])
```

VECTORIZING THE DATA

USING TFIDF VECTORIZATION for ngrams

In [ ]:

```
vectorizer1 = TfidfVectorizer(ngram_range = (1,1)) #ONLY
UNIGRAMS , #vectorizer1 = TfidfVectorizer(ngram_range = (2,2))
#ONLY BIGRAM , #vectorizer1 = TfidfVectorizer(ngram_range =
(1,2)) #UNI + BI ,

full_train_vectors=vectorizer1.fit_transform(full_train_twe
et) train_vectors=vectorizer1.transform(train_tweet)
val_vectors=vectorizer1.transform(val_tweet)
```

In [ ]:

```
test_vectors=vectorizer1.transform(test_tweet)
```

In [ ]:

```
train_vectors
```

Out[ ]:

```
<3074x8856 sparse matrix of type '<class 'numpy.float64'>'
        with 39804 stored elements in Compressed Sparse Row format>
```

for binary

In [ ]:

```
t1=train_df["task_
1"]
t2=val_df["task_1"
]
```

In [ ]:

```
t3=hasoc_test["task_1"]
```

for multiclass

In [ ]:

```
t4=train_df["task_
2"]
t5=val_df["task_2"
]
t6=hasoc_test["task_2"]
```

In [ ]:

```
from sklearn.metrics import classification_report
```

using OPTIMUM KERNEL AND PARAMETERS FOR SVM

In [ ]:

```
kernels = ['Polynomial', 'RBF', 'Sigmoid','Linear']#A function which returns the
corres ponding SVC model
def getClassifier(ktype):
    if ktype == 0:
        # Polynomial kernal
        return SVC(kernel='poly', degree=8, gamma="auto")
    elif ktype == 1:
        # Radial Basis Function kernal
        return SVC(kernel='rbf', gamma="auto")
    elif ktype == 2:
        # Sigmoid kernal
        return SVC(kernel='sigmoid', gamma="auto")
    elif ktype == 3:
        # Linear kernal
        return SVC(kernel='linear', gamma="auto")
```

FOR UNIGRAM ONLY

FOR BINARY CLASSIFICATION

PERFORMANCE ON ONLY HASOC DATA

VALIDATION 80:20 SPLIT

In [ ]:

```
for i in range(4):
    # Separate data into test and training sets
    svclassifier = getClassifier(i)
    svclassifier.fit(train_vectors,t1)# Make prediction
    y_pred = svclassifier.predict(val_vectors)# Evaluate our model
    print("Evaluation:", kernels[i],
    "kernel")
    print(classification_report(t2,y_pred))
```

```
Evaluation: Polynomial kernel
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       240
           1       0.69      1.00      0.82       529

    accuracy                           0.69       769
   macro avg       0.34      0.50      0.41       769
weighted avg       0.47      0.69      0.56       769
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
Evaluation: RBF kernel
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
Evaluation: Sigmoid kernel
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       240
           1       0.69      1.00      0.82       529

    accuracy                           0.69       769
   macro avg       0.34      0.50      0.41       769
weighted avg       0.47      0.69      0.56       769


/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

Evaluation: Linear kernel
              precision    recall  f1-score   support

           0       0.72      0.53      0.61       240
           1       0.81      0.91      0.86       529

    accuracy                           0.79       769
   macro avg       0.76      0.72      0.73       769
weighted avg       0.78      0.79      0.78       769
```

performance on test data

In [ ]:

```
for i in range(4):
    # Separate data into test and training sets
    svclassifier = getClassifier(i)
    svclassifier.fit(train_vectors,t1)# Make prediction
    y_pred = svclassifier.predict(test_vectors)# Evaluate our model
    print("Evaluation:", kernels[i],
    "kernel")
    print(classification_report(t3,y_pred))
```

```
Evaluation: Polynomial kernel
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       483
           1       0.62      1.00      0.77       798

    accuracy                           0.62      1281
   macro avg       0.31      0.50      0.38      1281
weighted avg       0.39      0.62      0.48      1281

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

Evaluation: RBF kernel
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
Evaluation: Sigmoid kernel
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       483
           1       0.62      1.00      0.77       798

    accuracy                           0.62      1281
   macro avg       0.31      0.50      0.38      1281
weighted avg       0.39      0.62      0.48      1281


/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

Evaluation: Linear kernel
```

FOR COMBINED (HASOC + KAGGLE DATA)

ON VALIDATION DATA

In [ ]:

```
for i in range(4):
    # Separate data into test and training sets
    svclassifier = getClassifier(i)
    svclassifier.fit(train_vectors,t1)# Make prediction
    y_pred = svclassifier.predict(val_vectors)# Evaluate our model
    print("Evaluation:", kernels[i],
    "kernel")
    print(classification_report(t2,y_pred))
```

```
Evaluation: Polynomial kernel
              precision    recall  f1-score   support

           0       0.87      1.00      0.93      6216
           1       0.00      0.00      0.00       945

    accuracy                           0.87      7161
   macro avg       0.43      0.50      0.46      7161
weighted avg       0.75      0.87      0.81      7161
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
Evaluation: RBF kernel
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
Evaluation: Sigmoid kernel
              precision    recall  f1-score   support

           0       0.87      1.00      0.93      6216
           1       0.00      0.00      0.00       945

    accuracy                           0.87      7161
   macro avg       0.43      0.50      0.46      7161
weighted avg       0.75      0.87      0.81      7161


/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

Evaluation: Linear kernel
```

ON TEST DATA

In [ ]:

```
for i in range(4):
    # Separate data into test and training sets
    svclassifier = getClassifier(i)
    svclassifier.fit(train_vectors,t1)# Make prediction
    y_pred = svclassifier.predict(test_vectors)# Evaluate our model
    print("Evaluation:", kernels[i],
    "kernel")
    print(classification_report(t3,y_pred))
```

```
Evaluation: Polynomial kernel
              precision    recall  f1-score   support

           0       0.38      1.00      0.55       483
           1       0.00      0.00      0.00       798

    accuracy                           0.38      1281
   macro avg       0.19      0.50      0.27      1281
weighted avg       0.14      0.38      0.21      1281

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

Evaluation: RBF kernel
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
Evaluation: Sigmoid kernel
              precision    recall  f1-score   support

           0       0.38      1.00      0.55       483
           1       0.00      0.00      0.00       798

    accuracy                           0.38      1281
   macro avg       0.19      0.50      0.27      1281
weighted avg       0.14      0.38      0.21      1281


/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

Evaluation: Linear kernel
```

FOR MULTICLASS CLASSIFICATION

FOR VALIDATION 80:20 SPLIT

In [ ]:

```
for i in range(4):
    # Separate data into test and training sets
    svclassifier = getClassifier(i)
    svclassifier.fit(train_vectors,t4)# Make prediction
    y_pred = svclassifier.predict(val_vectors)# Evaluate our model
    print("Evaluation:", kernels[i],
    "kernel")
    print(classification_report(t5,y_pred))
```

```
Evaluation: Polynomial kernel
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       141
           1       0.31      1.00      0.48       240
           2       0.00      0.00      0.00       140
           3       0.00      0.00      0.00       248

    accuracy                           0.31       769
   macro avg       0.08      0.25      0.12       769
weighted avg       0.10      0.31      0.15       769


/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

Evaluation: RBF kernel
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
Evaluation: Sigmoid kernel
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       141
           1       0.31      1.00      0.48       240
           2       0.00      0.00      0.00       140
           3       0.00      0.00      0.00       248

    accuracy                           0.31       769
   macro avg       0.08      0.25      0.12       769
weighted avg       0.10      0.31      0.15       769
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
Evaluation: Linear kernel
```

ON TEST DATA

In [ ]:

```
for i in range(4):
    # Separate data into test and training sets
    svclassifier = getClassifier(i)
    svclassifier.fit(train_vectors,t4)# Make prediction
    y_pred = svclassifier.predict(test_vectors)# Evaluate our model
    print("testing:", kernels[i],
    "kernel")
    print(classification_report(t6,y_pre
    d))
```

```
testing: Polynomial kernel
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       224
           1       0.38      1.00      0.55       483
           2       0.00      0.00      0.00       195
           3       0.00      0.00      0.00       379

    accuracy                           0.38      1281
   macro avg       0.09      0.25      0.14      1281
weighted avg       0.14      0.38      0.21      1281


/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

testing: RBF kernel
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       224
           1       0.38      1.00      0.55       483
           2       0.00      0.00      0.00       195
           3       0.00      0.00      0.00       379

    accuracy                           0.38      1281
   macro avg       0.09      0.25      0.14      1281
weighted avg       0.14      0.38      0.21      1281


/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
testing: Sigmoid kernel
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       224
           1       0.38      1.00      0.55       483
           2       0.00      0.00      0.00       195
           3       0.00      0.00      0.00       379

    accuracy                           0.38      1281
   macro avg       0.09      0.25      0.14      1281
weighted avg       0.14      0.38      0.21      1281


/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:
1308: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` pa
rameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

testing: Linear kernel
```

LSTM

Mount Drive

In [ ]:

```
from google.colab import
drive
drive.mount('/content/drive
')
```

Mounted at /content/drive

Importing Libraries and initializing stopwords and stemmer

In [ ]:

```
import numpy as np
import pandas as
pd
import matplotlib.pyplot as
plt import seaborn as sns

import warnings
warnings.filterwarnings("ignore")
import
nltk
import
re
from bs4 import BeautifulSoup
from tqdm import tqdm
from nltk.stem import WordNetLemmatizer

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,f1_score, confusion_matrix

from keras.preprocessing.text import Tokenizer
from sklearn.feature_extraction.text import TfidfVectorizer
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Dense , Input , LSTM , Embedding, Dropout , Activation,
GRU, F latten
from keras.layers import Bidirectional, GlobalMaxPool1D
from keras.models import Model, Sequential
from keras.layers import Convolution1D
from keras import initializers, regularizers, constraints, optimizers,
layers nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
```

Out[ ]:

True

In [ ]:

```
from sklearn.preprocessing import
LabelEncoder
```

Reading Data

In [ ]:

```
hasoc_train=pd.read_csv("/content/drive/MyDrive/NLP
 PROJECT/en_Hasoc2021_train.csv") # hasoc only data
# combined data of hasoc+kaggle
hasoc_test=pd.read_csv("/content/drive/MyDrive/NLP
PROJECT/en_Hasoc2021_test_data.csv")
# hasoc test data
hasoc_test_bin=pd.read_csv("/content/drive/MyDrive/NLP
PROJECT/1A_English_actual_label s.csv") # hasoc binary test labels
hasoc_test_mul=pd.read_csv("/content/drive/MyDrive/NLP
PROJECT/1B_English_actual_label s.csv") #hasoc multiclass test labels
```

In [ ]:

```
combined_train=pd.read_csv("/content/drive/MyDrive/NLP
PROJECT/hate_bin_combine.csv",en coding = "ISO-8859-1")
```

In [ ]:

```
hasoc_train
```

Out[ ]:

	Unnamed: 0	_id	text	task_1	task_2
0	4986	60c5d6bf5659ea5e55defa2c	@wealth if you made it through this &...	HOF	PRFN
1	3394	60c5d6bf5659ea5e55def461	Technically that's still turning back the cloc...	HOF	OFFN
2	1310	60c5d6bf5659ea5e55defaad	@VMBJP @BJP4Bengal @BJP4India @narendramodi @J...	NOT	NONE
3	3390	60c5d6bf5659ea5e55def419	@krtoprak_yigit Soldier of Japan Who has dick ...	HOF	OFFN
4	4626	60c5d6bf5659ea5e55def7fa	@blueheartedly You'd be better off asking who ...	HOF	OFFN
...	...	...	...	...	...
3838	1661	60c5d6bf5659ea5e55defd57	@BBCNews Let the dog deal with the wanker once...	HOF	PRFN
3839	194	60c5d6bf5659ea5e55def185	India has suffered a lot. That Chinese bastard...	HOF	HATE
3840	3988	60c5d6bf5659ea5e55def78c	People didn't give 300+ seats majority to BJP ...	HOF	HATE
3841	4212	60c5d6bf5659ea5e55defb04	@KanganaTeam This is such a vile, xenophobic a...	HOF	PRFN
3842	1512	60c5d6bf5659ea5e55defb8a	@30iPpgStmILw0SI @ChinaDaily #ChineseVirus #Wu...	NOT	NONE

3843 rows × 5 columns

In [ ]:

```
combined_train
```

Out[ ]:

	_id	text	task_1
0	60c5d6bf5659ea5e55defa2c	@wealth if you made it through this &...	HOF
1	60c5d6bf5659ea5e55def461	Technically that's still turning back the cloc...	HOF
2	60c5d6bf5659ea5e55defaad	@VMBJP @BJP4Bengal @BJP4India @narendramodi @J...	NOT
3	60c5d6bf5659ea5e55def419	@krtoprak_yigit Soldier of Japan Who has dick ...	HOF
4	60c5d6bf5659ea5e55def7fa	@blueheartedly You'd be better off asking who ...	HOF
...	...	...	...
35800	31958	ate @user isz that youuu? Ã Â°Ã Â Ã Â Ã Â Ã Â Ã Â°Ã...	NOT
35801	31959	to see nina turner on the airwaves trying to w...	NOT
35802	31960	listening to sad songs on a monday morning otw...	NOT
35803	31961	@user #sikh #temple vandalised in in #calgary,...	HOF
35804	31962	thank you @user for you follow	NOT

35805 rows × 3 columns

In [ ]:

```
hasoc_test["task_1"]=hasoc_test_bin["label"]
```

In [ ]:

```
hasoc_test["task_2"]=hasoc_test_mul["label"]
```

Basic Preprocessing of Tweets

In [ ]:

```
lemmatizer = WordNetLemmatizer()
stop_words= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselve
s', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 't
hey', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
at'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
d', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as'
, 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through'
, 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'ov
er', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
y', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too'
, 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
w', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'migh
tn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
asn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't","not"])
```

In [ ]:

```
def preprocess(tweets):

  tweets.encode('ascii', 'ignore').decode('ascii') #remove
  emojis tweets=tweets.lower() #convert to lower case tweets =
  re.sub(r"http\S+", " ", tweets) #remove urls
  l=tweets.split(" ")
  for n,i in enumerate(l):
    if '@' in i:
      l[n]=""

  tweets=" ".join(l)
  tweets= re.sub('[^A-Za-z]+',' ',tweets) #remove bad character
  tweets = [word for word in tweets.split(" ") if not word in stop_words] # removing
st op words
  tweets= [lemmatizer.lemmatize(token, "v") for token in tweets] #Lemmatization
  tweets=" ".join(tweets)
  return tweets
def decontract(text):
    text = re.sub(r"won\'t", "will not",
    text) text = re.sub(r"can\'t", "can
    not", text) text = re.sub(r"n\'t", "
    not", text)
    text = re.sub(r"\'re", " are", text)
    text = re.sub(r"\'s", " is", text)
    text = re.sub(r"\'d", " would", text)
    text = re.sub(r"\'ll", " will", text)
    text = re.sub(r"\'t", " not", text)
    text = re.sub(r"\'ve", " have", text)
    text = re.sub(r"\'m", " am", text)
    return text
```

when using only hasoc

In [ ]:

```
hasoc_train["text"]=hasoc_train["text"].apply(lambda
x:decontract(x))
hasoc_train["text"]=hasoc_train["text"].apply(lambda
x:preprocess(x))
```

In [ ]:

```
train_df, val_df = train_test_split(hasoc_train, test_size = 0.2, random_state =
42) train_tweet=train_df.text
val_tweet=val_df.text
```

when using combined data

In [ ]:

```
combined_train["text"]=combined_train["text"].apply(lambda
x:decontract(x))
combined_train["text"]=combined_train["text"].apply(lambda
x:preprocess(x))
```

In [ ]:

```
train_df, val_df = train_test_split(combined_train, test_size = 0.2, random_state =
42) train_tweet=train_df.text
val_tweet=val_df.text
```

preprocessing the testdata

In [ ]:

```
hasoc_test["text"]=hasoc_test["text"].apply(lambda
x:decontract(x))
hasoc_test["text"]=hasoc_test["text"].apply(lambda
x:preprocess(x))
```

In [ ]:

```
test_tweet=hasoc_test.text
```

In [ ]:

```
le=LabelEncoder()
```

on evaluation data

for binary data

In [ ]:

```
train_df["task_1"] =
le.fit_transform(train_df["task_1"]) val_df["task_1"]
= le.fit_transform(val_df["task_1"])
```

In [ ]:

```
hasoc_test["task_1"]=le.fit_transform(hasoc_test["task_1"])
```

for multiclass data

In [ ]:

```
train_df["task_2"] =
le.fit_transform(train_df["task_2"]) val_df["task_2"]
= le.fit_transform(val_df["task_2"])
```

In [ ]:

```
hasoc_test["task_2"]=le.fit_transform(hasoc_test["task_2"])
```

run for both

In [ ]:

```
full_train_tweet=hasoc_train["text"]
```

In [ ]:

```
full_train_tweet=combined_train["text"] # run this only for combined data
```

In [ ]:

```
train_tweet=train_df["text"]
```

In [ ]:

```
val_tweet=val_df["text"]
```

FOR BINARY

In [ ]:

```
t1=train_df["task_
1"]
t2=val_df["task_1"
]
```

In [ ]:

```
t3=hasoc_test["task_1"]
```

Basic LSTM model and training

In [ ]:

```
top words =         #top 6000 words in the
max_review_length = 130 #maximum sentence length
embedding_vector_length = 32   #each word is mapped to a 32 dimensional vector
tokenizer = Tokenizer(num_words=top_words) # top 6000 are getting tokenized
tokenizer.fit_on_texts(train_tweet)
list_tokenized_train = tokenizer.texts_to_sequences(train_tweet) # text data is
convert ed to sequence data
```

run this for binary only

In [ ]:

```
X_train = pad_sequences(list_tokenized_train, maxlen=max_review_length) # making all i
nput sequence into same length
y_train = t1
```

```
model = Sequential()
model.add(Embedding(top_words+1, embedding_vector_length,
input_length=max_review_lengt h))
model.add(LSTM(100))
model.add(Dense(1, activation='relu'))
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy']) model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_4 (Embedding)      (None, 130, 32)           192032

 lstm_4 (LSTM)                (None, 100)               53200

 dense_4 (Dense)              (None, 1)                 101

=================================================================
Total params: 245,333
Trainable params: 245,333
Non-trainable params: 0
_________________________________________________________________
```

for binary classification

In [ ]:

```
history = model.fit(X_train,y_train, epochs=30, batch_size=64)
```

```
Epoch 1/30
49/49 [==============================] - 9s 148ms/step - loss: 0.7730 - ac
curacy: 0.6295
Epoch 2/30
49/49 [==============================] - 7s 148ms/step - loss: 0.5313 - ac
curacy: 0.7115
Epoch 3/30
49/49 [==============================] - 7s 146ms/step - loss: 0.3840 - ac
curacy: 0.8679
Epoch 4/30
49/49 [==============================] - 7s 148ms/step - loss: 0.4747 - ac
curacy: 0.8774
Epoch 5/30
49/49 [==============================] - 7s 149ms/step - loss: 0.2045 - ac
curacy: 0.9258
Epoch 6/30
49/49 [==============================] - 7s 147ms/step - loss: 0.1471 - ac
curacy: 0.9486
Epoch 7/30
49/49 [==============================] - 7s 148ms/step - loss: 0.1217 - ac
curacy: 0.9655
Epoch 8/30
49/49 [==============================] - 7s 149ms/step - loss: 0.1048 - ac
curacy: 0.9736
Epoch 9/30
49/49 [==============================] - 7s 149ms/step - loss: 0.2009 - ac
curacy: 0.9525
Epoch 10/30
49/49 [==============================] - 7s 149ms/step - loss: 0.1220 - ac
curacy: 0.9750
Epoch 11/30
49/49 [==============================] - 7s 150ms/step - loss: 0.0936 - ac
curacy: 0.9828
Epoch 12/30
49/49 [==============================] - 7s 148ms/step - loss: 0.0930 - ac
curacy: 0.9834
Epoch 13/30
49/49 [==============================] - 7s 149ms/step - loss: 0.1030 - ac
curacy: 0.9867
Epoch 14/30
49/49 [==============================] - 7s 150ms/step - loss: 0.0933 - ac
curacy: 0.9873
Epoch 15/30
49/49 [==============================] - 7s 149ms/step - loss: 0.1037 - ac
curacy: 0.9886
Epoch 16/30
49/49 [==============================] - 7s 147ms/step - loss: 0.1080 - ac
curacy: 0.9896
Epoch 17/30
49/49 [==============================] - 7s 147ms/step - loss: 0.1057 - ac
curacy: 0.9893
Epoch 18/30
49/49 [==============================] - 7s 147ms/step - loss: 0.1050 - ac
curacy: 0.9909
Epoch 19/30
49/49 [==============================] - 7s 147ms/step - loss: 0.1039 - ac
curacy: 0.9886
Epoch 20/30
49/49 [==============================] - 7s 148ms/step - loss: 0.1140 - ac
curacy: 0.9912
Epoch 21/30
```

```
49/49 [==============================] - 7s 147ms/step - loss: 1.8884 - ac
curacy: 0.7895
Epoch 22/30
49/49 [==============================] - 7s 148ms/step - loss: 0.7542 - ac
curacy: 0.8569
Epoch 23/30
49/49 [==============================] - 7s 147ms/step - loss: 0.6971 - ac
curacy: 0.8725
Epoch 24/30
49/49 [==============================] - 7s 149ms/step - loss: 0.4160 - ac
curacy: 0.9265
Epoch 25/30
49/49 [==============================] - 7s 146ms/step - loss: 0.3031 - ac
curacy: 0.9541
Epoch 26/30
49/49 [==============================] - 7s 146ms/step - loss: 0.2497 - ac
curacy: 0.9655
Epoch 27/30
49/49 [==============================] - 7s 146ms/step - loss: 0.2196 - ac
curacy: 0.9707
Epoch 28/30
49/49 [==============================] - 7s 147ms/step - loss: 0.2075 - ac
curacy: 0.9736
Epoch 29/30
49/49 [==============================] - 7s 147ms/step - loss: 0.1718 - ac
curacy: 0.9740
Epoch 30/30
49/49 [==============================] - 7s 147ms/step - loss: 0.1467 - ac
curacy: 0.9753
```

On Evaluation data

In [ ]:

```
list_tokenized_test = tokenizer.texts_to_sequences(val_df['text'])
X_test = pad_sequences(list_tokenized_test,
maxlen=max_review_length) y_test = t2
prediction =
model.predict(X_test) y_pred =
(prediction > 0.5)
print("Accuracy of the model : ", accuracy_score(y_pred,
y_test)) print('F1-score: ', f1_score(y_pred, y_test))
print('Confusion matrix:')
confusion_matrix(y_test,y_pred)
```

```
Accuracy of the model : 0.7581274382314694
F1-score: 0.6172839506172839
Confusion matrix:
```

Out[ ]:

```
array([[433, 96],
       [ 90, 150]])
```

In [ ]:

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.82      0.82       529
           1       0.61      0.62      0.62       240

    accuracy                           0.76       769
   macro avg       0.72      0.72      0.72       769
weighted avg       0.76      0.76      0.76       769
```

on test data

In [ ]:

```
list_tokenized_test =
tokenizer.texts_to_sequences(hasoc_test['text']) X_test =
pad_sequences(list_tokenized_test, maxlen=max_review_length) y_test
= t3
prediction =
model.predict(X_test) y_pred =
(prediction > 0.5)
print("Accuracy of the model : ", accuracy_score(y_pred,
y_test)) print('F1-score: ', f1_score(y_pred, y_test))
print('Confusion matrix:')
confusion_matrix(y_test,y_pred)
```

```
Accuracy of the model : 0.7244340359094458
F1-score: 0.6175514626218851
Confusion matrix:
```

Out[ ]:

```
array([[643, 155],
       [198, 285]])
```

In [ ]:

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.76      0.81      0.78       798
           1       0.65      0.59      0.62       483

    accuracy                           0.72      1281
   macro avg       0.71      0.70      0.70      1281
weighted avg       0.72      0.72      0.72      1281
```

run this for Combined only

In [ ]:

```
X_train = pad_sequences(list_tokenized_train,
maxlen=max_review_length) y_train = t1
```

```
model = Sequential()
model.add(Embedding(top_words+1, embedding_vector_length,
input_length=max_review_lengt h))
model.add(LSTM(100))
model.add(Dense(1, activation='relu'))
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy']) model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_3 (Embedding)      (None, 130, 32)           192032

 lstm_3 (LSTM)                (None, 100)               53200

 dense_3 (Dense)              (None, 1)                 101

=================================================================
Total params: 245,333
Trainable params: 245,333
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
history = model.fit(X_train,y_train, epochs=30, batch_size=64)
```

```
Epoch 1/30
448/448 [==============================] - 70s 150ms/step - loss: 2.0482 -
accuracy: 0.8495
Epoch 2/30
448/448 [==============================] - 67s 150ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 3/30
448/448 [==============================] - 68s 151ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 4/30
448/448 [==============================] - 68s 152ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 5/30
448/448 [==============================] - 68s 152ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 6/30
448/448 [==============================] - 68s 151ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 7/30
448/448 [==============================] - 67s 150ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 8/30
448/448 [==============================] - 67s 150ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 9/30
448/448 [==============================] - 68s 151ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 10/30
448/448 [==============================] - 67s 151ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 11/30
448/448 [==============================] - 67s 150ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 12/30
448/448 [==============================] - 67s 149ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 13/30
448/448 [==============================] - 67s 149ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 14/30
448/448 [==============================] - 67s 149ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 15/30
448/448 [==============================] - 67s 150ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 16/30
448/448 [==============================] - 66s 148ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 17/30
448/448 [==============================] - 67s 149ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 18/30
448/448 [==============================] - 67s 150ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 19/30
448/448 [==============================] - 67s 149ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 20/30
448/448 [==============================] - 67s 148ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 21/30
```

```
448/448 [==============================] - 66s 148ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 22/30
448/448 [==============================] - 67s 149ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 23/30
448/448 [==============================] - 67s 149ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 24/30
448/448 [==============================] - 67s 149ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 25/30
448/448 [==============================] - 67s 149ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 26/30
448/448 [==============================] - 67s 149ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 27/30
448/448 [==============================] - 67s 150ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 28/30
448/448 [==============================] - 67s 150ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 29/30
448/448 [==============================] - 67s 149ms/step - loss: 2.0219 -
accuracy: 0.8674
Epoch 30/30
448/448 [==============================] - 67s 149ms/step - loss: 2.0219 -
accuracy: 0.8674
```

On Evaluation data

In [ ]:

```
list_tokenized_test = tokenizer.texts_to_sequences(val_df['text'])
X_test = pad_sequences(list_tokenized_test,
maxlen=max_review_length) y_test = t2
prediction =
model.predict(X_test) y_pred =
(prediction > 0.5)
print("Accuracy of the model : ", accuracy_score(y_pred,
y_test)) print('F1-score: ', f1_score(y_pred, y_test))
print('Confusion matrix:')
confusion_matrix(y_test,y_pred)
```

```
Accuracy of the model : 0.8680351906158358
F1-score:  0.9293563579277865
Confusion matrix:
```

Out[ ]:

```
array([[   0, 945],
       [   0, 6216]])
```

In [ ]:

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       945
           1       0.87      1.00      0.93      6216

    accuracy                           0.87      7161
   macro avg       0.43      0.50      0.46      7161
weighted avg       0.75      0.87      0.81      7161
```

on test data

In [ ]:

```
list_tokenized_test =
tokenizer.texts_to_sequences(hasoc_test['text']) X_test =
pad_sequences(list_tokenized_test, maxlen=max_review_length) y_test
= t3
prediction =
model.predict(X_test) y_pred =
(prediction > 0.5)
print("Accuracy of the model : ", accuracy_score(y_pred,
y_test)) print('F1-score: ', f1_score(y_pred, y_test))
print('Confusion matrix:')
confusion_matrix(y_test,y_pred)
```

```
Accuracy of the model : 0.3770491803278688
F1-score:  0.5476190476190476
Confusion matrix:
```

Out[ ]:

```
array([[  0, 798],
       [  0, 483]])
```

In [ ]:

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       798
           1       0.38      1.00      0.55       483

    accuracy                           0.38      1281
   macro avg       0.19      0.50      0.27      1281
weighted avg       0.14      0.38      0.21      1281
```

DistilBert

Mount Drive

In [ ]:

```
from google.colab import
drive
drive.mount('/content/drive
')
```

```
Mounted at /content/drive
```

Importing Libraries and initializing stopwords and stemmer

In [ ]:

```
!pip3 install texthero
!pip3 install transformers
!pip3 install tensorflow_addons
```

In [ ]:

```
!pip install tweet-preprocessor

import seaborn as sns
import matplotlib.pyplot as
plt import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
Collecting tweet-preprocessor
  Downloading tweet_preprocessor-0.6.0-py3-none-any.whl (27 kB)
Installing collected packages: tweet-preprocessor
Successfully installed tweet-preprocessor-0.6.0
```

In [ ]:

```
import re
import nltk
from wordcloud import WordCloud
from nltk.stem import WordNetLemmatizer
from textblob import TextBlob,Word
from nltk.corpus import
words
nltk.download('words')
nltk.download('wordnet')
import texthero as hero # to be learned
import re
from texthero import stopwords

from nltk.corpus import wordnet #to be learned

import tensorflow as tf

from nltk.corpus import stopwords
from nltk.tokenize import TweetTokenizer

import tensorflow as tf

from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score,f1_score, confusion_matrix
```

```
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Unzipping corpora/words.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

In [ ]:

```
def lemma_per_pos(sent):
    '''function to lemmatize according to part of speech tag'''
    tweet_tokenizer=TweetTokenizer()
    lemmatizer = nltk.stem.WordNetLemmatizer()
    lemmatized_list = [lemmatizer.lemmatize(w) for w in tweet_tokenizer.tokenize(sent
)]
    return " ".join(lemmatized_list)

def df_preprocessing(df,feature_col):
    '''
    Preprocessing of
    dataframe '''
    stop = set(stopwords.words('english'))
    df[feature_col]= (df[feature_col].pipe(hero.lowercase).
                      pipe(hero.remove_urls).
                      pipe(hero.remove_digits).
                      pipe(hero.remove_punctuatio
                      n).
                      pipe(hero.remove_html_tags)
                      )
    # Lemmatization
    df[feature_col]= [lemma_per_pos(sent) for sent in
    df[feature_col]] #df[col_name]=
    hero.remove_stopwords(df[col_name],custom_stopwords) return df
```

In [ ]:

```
from transformers import AutoTokenizer,TFDistilBertModel, DistilBertConfig
from transformers import TFAutoModel
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Dropout
from transformers import AdamW, get_linear_schedule_with_warmup
import tensorflow_addons as tfa
```

Reading Data

In [ ]:

```
hasoc_train = pd.read_csv("/content/drive/MyDrive/NLP
PROJECT/modified_hasoc_task21.cs v") #Your respective address
hasoc_train.head()
## change data
```

Out[ ]:

	_id	text	hate	offen	prof	none
0	60c5d6bf5659ea5e55defa2c	@wealth if you made it through this &...	0	0	1	0
1	60c5d6bf5659ea5e55def461	Technically that's still turning back the cloc...	0	1	0	0
2	60c5d6bf5659ea5e55defaad	@VMBJP @BJP4Bengal @BJP4India @narendramodi @J...	0	0	0	1
3	60c5d6bf5659ea5e55def419	@krtoprak_yigit Soldier of Japan Who has dick ...	0	1	0	0
4	60c5d6bf5659ea5e55def7fa	@blueheartedly You'd be better off asking who ...	0	1	0	0

In [ ]:

```
hasoc_train.drop("_id",inplace=True,axis=1)
```

In [ ]:

```
hasoc_test = pd.read_csv("/content/drive/MyDrive/NLP
PROJECT/en_Hasoc2021_test_data.cs v")
hasoc_test.head()
```

Out[ ]:

	_id	text
0	60c5d6bf5659ea5e55deffcb	Fewer people coming in for vaccinations. So sa...
1	60c5d6bf5659ea5e55df028c	@MattHancock This may all be true. But... What...
2	60c5d6bf5659ea5e55def377	@Layla_EFC I've unfollowed him the wanker
3	60c5d6bf5659ea5e55def4c7	You guys are losing it all over the world. The...
4	60c5d6bf5659ea5e55df01a6	And thus death laughs... It is sad merriment, ...

In [ ]:

```
hasoc_test.drop("_id",inplace=True,axis=1)
```

In [ ]:

```
combined_data=pd.read_csv("/content/drive/MyDrive/NLP
PROJECT/hate_bin_combine.csv",enc oding="ISO-8859-1")
```

In [ ]:

```
combined_data.head()
```

Out[ ]:

	_id	text	task_1
0	60c5d6bf5659ea5e55defa2c	@wealth if you made it through this &...	HOF
1	60c5d6bf5659ea5e55def461	Technically that's still turning back the cloc...	HOF
2	60c5d6bf5659ea5e55defaad	@VMBJP @BJP4Bengal @BJP4India @narendramodi @J...	NOT
3	60c5d6bf5659ea5e55def419	@krtoprak_yigit Soldier of Japan Who has dick ...	HOF
4	60c5d6bf5659ea5e55def7fa	@blueheartedly You'd be better off asking who ...	HOF

In [ ]:

```
combined_data.drop("_id",inplace=True,axis=1)
```

In [ ]:

```
test_bin_labels=pd.read_csv("/content/drive/MyDrive/NLP
PROJECT/1A_English_actual_label s.csv")
test_mul_labels=pd.read_csv("/content/drive/MyDrive/NLP
PROJECT/1B_English_actual_label s.csv")
```

run when performing multiclass

In [ ]:

```
hasoc_test["task_2"]=test_mul_labels["label"]
```

run when performing binary class

In [ ]:

```
hasoc_test["task_1"]=test_bin_labels["label"]
```

ONE HOT ENCODING CLASS LABELS

In [ ]:

```
def one_hot(data):
  column_names =
  ["text","hate","offen","prof","none"] test_df =
  pd.DataFrame(columns = column_names)

  for i in range(len(data)):
    text_temp = data['text'].iloc[i]
    if data['task_2'].iloc[i] ==
      "NONE": test_df.loc[i] =
      [text_temp,0,0,0,1]
    elif data['task_2'].iloc[i] ==
      "PRFN": test_df.loc[i] =
      [text_temp,0,0,1,0]
    elif data['task_2'].iloc[i] ==
      "OFFN": test_df.loc[i] =
      [text_temp,0,1,0,0]
    elif data['task_2'].iloc[i] ==
      "HATE": test_df.loc[i] =
      [text_temp,1,0,0,0]
  return test_df
```

In [ ]:

```
hasoc_test=one_hot(hasoc_test)
```

In [ ]:

```
hasoc_train
```

Out[ ]:

	text	hate	offen	prof	none
0	@wealth if you made it through this &...	0	0	1	0
1	Technically that's still turning back the cloc...	0	1	0	0
2	@VMBJP @BJP4Bengal @BJP4India @narendramodi @J...	0	0	0	1
3	@krtoprak_yigit Soldier of Japan Who has dick ...	0	1	0	0
4	@blueheartedly You'd be better off asking who ...	0	1	0	0
...	...	...	...	...	...
3838	@BBCNews Let the dog deal with the wanker once...	0	0	1	0
3839	India has suffered a lot. That Chinese bastard...	1	0	0	0
3840	People didn't give 300+ seats majority to BJP ...	1	0	0	0
3841	@KanganaTeam This is such a vile, xenophobic a...	0	0	1	0
3842	@30iPpgStmILw0SI @ChinaDaily #ChineseVirus #Wu...	0	0	0	1

3843 rows × 5 columns

In [ ]:

```
target_col= hasoc_train.columns[1:]
feature_col=hasoc_train.columns[0:1]
```

Preprocessing

In [ ]:

```
target_col,feature_col
```

Out[ ]:

```
(Index(['hate', 'offen', 'prof', 'none'], dtype='object'),
 Index(['text'], dtype='object'))
```

preprocessing training and testing data

In [ ]:

```
with tf.device('/GPU:0'):
  hasoc_tr_prp= df_preprocessing(hasoc_train,feature_col[0])
```

In [ ]:

```
hasoc_tt_prp = df_preprocessing(hasoc_test,feature_col[0])
```

In [ ]:

```
#Creating tokenizer
def create_tokenizer(pretrained_weights='distilbert-base-uncased'):
  '''Function to create the tokenizer'''

  tokenizer = AutoTokenizer.from_pretrained(pretrained_weights)
  return tokenizer

#Tokenization of the data
def data_tokenization(dataset,feature_col,max_len,tokenizer):
    '''dataset: Pandas dataframe with feature name is column
    name Pretrained_weights: selected model
    RETURN: [input_ids, attention_mask]'''

    tokens = dataset[feature_col].apply(lambda x: tokenizer(x,return_tensors='tf',
                                                            truncation=True,
                                                            padding='max_lengt
                                                            h',
                                                            max_length=max_len
                                                            ,
                                                            add_special_tokens=True))
    input_ids= []
    attention_mask=[]
    for item in tokens:
        input_ids.append(item['input_ids'])
        attention_mask.append(item['attention_mask'])
    input_ids, attention_mask=np.squeeze(input_ids), np.squeeze(attention_mask)

    return [input_ids,attention_mask]
```

Model

In [ ]:

```
def bert_model(pretrained_weights,max_len,learning_rate):
  '''BERT model creation with pretrained weights
  INPUT:
  pretrained_weights: Language model pretrained weights
  max_len: input length '''
  print('Model selected:', pretrained_weights)
  bert=TFAutoModel.from_pretrained(pretrained_weights)

  # This is must if you would like to train the layers of language models too.
  for layer in bert.layers:
      layer.trainable = True

  # # parameter declaration
  # step = tf.Variable(0, trainable=False)
  # schedule = tf.optimizers.schedules.PiecewiseConstantDecay([10000, 15000], [2e-0, 2e
-1, 1e-2])
  # # lr and wd can be a function or a tensor
  # lr = learning_rate * schedule(step)
  # wd = lambda:lr * schedule(step)
  # optimizer = tfa.optimizers.AdamW(learning_rate=lr, weight_decay=wd)

  # optimizer= tf.keras.optimizers.Adam(learning_rate=learning_rate, beta_1=0.9, beta_2
=0.999, epsilon=1e-07, amsgrad=False,name='Adam')
  optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate)

  # declaring inputs, BERT take input_ids and attention_mask as input
  input_ids= Input(shape=(max_len,),dtype=tf.int32,name='input_ids')
  attention_mask=Input(shape=(max_len,),dtype=tf.int32,name='attention_mask')

  bert= bert(input_ids,attention_mask=attention_mask)
  x= bert[0][:,0,:]
  x=tf.keras.layers.Dropout(0.05)(x)
  # x= tf.keras.layers.Dense(128)(x)
  x=tf.keras.layers.Dense(64)(x)
  x=tf.keras.layers.Dense(32)(x)

  output=tf.keras.layers.Dense(4,activation='sigmoid')(x)

  model=Model(inputs=[input_ids,attention_mask],outputs=[output])
  # compiling model
  model.compile(optimizer=optimizer,
                loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True, reductio
n=tf.keras.losses.Reduction.NONE,name='categorical_crossentropy'),
                metrics=['accuracy'])
  return model
```

In [ ]:

```
pretrained_weights='bert-base-uncas
ed' max_len=256
epochs=6
learning_rate=2
e-5
batch_size=4
```

In [ ]:

```
tokenizer= create_tokenizer()
```

```
Downloading:   0%|          | 0.00/28.0 [00:00<?, ?B/s]

Downloading:   0%|          | 0.00/570 [00:00<?, ?B/s]

Downloading:   0%|          | 0.00/226k [00:00<?,

?B/s] Downloading:          0%|     | 0.00/455k

[00:00<?, ?B/s]
```

In [ ]:

```
x_train= data_tokenization(hasoc_tr_prp,feature_col[0],max_len,tokenizer)
```

In [ ]:

```
y_train=
hasoc_tr_prp[target_col].values
y_train
```

Out[ ]:

```
array([[0, 0, 1, 0],
       [0, 1, 0, 0],
       [0, 0, 0, 1],
       ...,
       [1, 0, 0, 0],
       [0, 0, 1, 0],
       [0, 0, 0, 1]])
```

In [ ]:

```
y_train.shape
```

Out[ ]:

```
(3843, 4)
```

In [ ]:

```
bert=bert_model(pretrained_weights,max_len,learning_ra
te) bert.summary()
```

```
Model selected:
bert-base-uncased
```

Some layers from the model checkpoint at bert-base-uncased were not used w
hen initializing TFBertModel: ['nsp_cls', 'mlm_cls']

- This IS expected if you are initializing TFBertModel from the checkpoint of a model trained on another task or with another architecture (e.g. init ializing a BertForSequenceClassification model from a BertForPreTraining m odel).
- This IS NOT expected if you are initializing TFBertModel from the checkp oint of a model that you expect to be exactly identical (initializing a Be rtForSequenceClassification model from a BertForSequenceClassification mod el).

All the layers of TFBertModel were initialized from the model checkpoint a
t bert-base-uncased.

If your task is similar to the task the model of the checkpoint was traine
d on, you can already use TFBertModel for predictions without further trai
ning.

```
Model:
 "model_1"
```

```
 Layer (type)                   Output Shape         Param #     Connected
to
==================================================================================
=======================
 input_ids (InputLayer)         [(None, 256)]        0           []

 attention_mask (InputLayer)    [(None, 256)]        0           []

 tf_bert_model_1 (TFBertModel)  TFBaseModelOutputWi  109482240   ['input_i
ds[0][0]',
                                thPoolingAndCrossAt               'attenti
on_mask[0][0]']
                                tentions(last_hidde
                                n_state=(None, 256,
                                 768),
                                 pooler_output=(Non
                                e, 768),
                                 past_key_values=No
                                ne, hidden_states=N
                                one, attentions=Non
                                e, cross_attentions
                                =None)

 tf._operators_.getitem_1   (Sl  (None, 768)         0           ['tf_bert
_model_1[0][0]']
 icingOpLambda)

 dropout_75 (Dropout)           (None, 768)          0           ['tf._op
erators_.getitem_1[0][0
                                                                 ]']

 dense_3 (Dense)                (None, 64)           49216       ['dropout
_75[0][0]']

 dense_4 (Dense)                (None, 32)           2080        ['dense_3
[0][0]']

 dense_5 (Dense)                (None, 4)            132         ['dense_4
[0][0]']

==================================================================================
=======================
Total params: 109,533,668
Trainable params: 109,533,668
Non-trainable params: 0
```

In [ ]:

```
x_train=x_train.astype(np.float32)
y_train=x_train.asarray(y_train).astype(np.float32)
```

In [ ]:

```
with tf.device('/GPU:0'):
    bert.fit(x_train,y_train,batch_size=batch_size,epochs=epochs,verbose=1)
```

```
Epoch 1/6

/usr/local/lib/python3.7/dist-packages/tensorflow/python/util/dispatch.py:
1096: UserWarning:

"`binary_crossentropy` received `from_logits=True`, but the `output` argum
ent was produced by a sigmoid or softmax activation and thus does not repr
esent logits. Was this intended?"


WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model_1/
bert/pooler/dense/kernel:0', 'tf_bert_model_1/bert/pooler/dense/bias:0'] w
hen minimizing the loss. If you're using `model.compile()`, did you forget
to provide a `loss`argument?
WARNING:tensorflow:Gradients do not exist for variables ['tf_bert_model_1/
bert/pooler/dense/kernel:0', 'tf_bert_model_1/bert/pooler/dense/bias:0'] w
hen minimizing the loss. If you're using `model.compile()`, did you forget
to provide a `loss`argument?
961/961 [==============================] - 489s 487ms/step - loss: 0.4134
-     accuracy:
0.6071 Epoch 2/6
961/961 [==============================] - 468s 487ms/step - loss: 0.2987
-     accuracy:
0.7335 Epoch 3/6
961/961 [==============================] - 467s 486ms/step - loss: 0.1844
-     accuracy:
0.8525 Epoch 4/6
961/961 [==============================] - 468s 487ms/step - loss: 0.0874
-     accuracy:
0.9394 Epoch 5/6
961/961 [==============================] - 466s 485ms/step - loss: 0.0531
-     accuracy:
0.9649 Epoch 6/6
961/961 [==============================] - 465s 483ms/step - loss: 0.0421
- accuracy: 0.9727
```

In [ ]:

```
!mkdir -p saved_model
bert.save('saved_model/my_model')
```

```
WARNING:absl:Found untraced functions such as embeddings_layer_call_fn, em
beddings_layer_call_and_return_conditional_losses, encoder_layer_call_fn,
encoder_layer_call_and_return_conditional_losses, pooler_layer_call_fn whi
le saving (showing 5 of 1055). These functions will not be directly callab
le after loading.
INFO:tensorflow:Assets written to: saved_model/my_model/assets

INFO:tensorflow:Assets written to: saved_model/my_model/assets

/usr/local/lib/python3.7/dist-packages/keras/saving/saved_model/layer_seri
alization.py:112: CustomMaskWarning:

Custom mask layers require a config and must override get_config. When loa
ding, the custom mask layer must be passed to the custom_objects argument.
```

tokenizing test data

In [ ]:

```
x_test=
data_tokenization(hasoc_tt_prp,feature_col[0],max_len,tokenizer)
x_test
```

Out[ ]:

```
[array([[  101,  8491,  2111  ...,     0,     0,     0],
                        ,
        [  101,  4717,  4819  ...,     0,     0,     0],
                        ,
        [  101, 19786,  1041  ...,     0,     0,     0],
                        ,
        ...,
        [  101, 22079,  2050  ...,     0,     0,     0],
                        ,
        [  101,  2282, 21759  ...,     0,     0,     0],
                        ,
        [  101, 15467, 19722  ...,     0,     0,     0]], dtype=int32),
                        ,
 array([[1, 1, 1, ..., 0, 0, 0],
        [1, 1, 1, ..., 0, 0, 0],
        [1, 1, 1, ..., 0, 0, 0],
        ...,
        [1, 1, 1, ..., 0, 0, 0],
        [1, 1, 1, ..., 0, 0, 0],
        [1, 1, 1, ..., 0, 0, 0]], dtype=int32)]
```

In [ ]:

```
preds= bert.predict(x_test)
```

In [ ]:

```
preds.shape
predicted_test_labels=pd.DataFrame(preds)
```

In [ ]:

```
predicted_test_labels
```

Out[ ]:

	hate	offen	prof	none
0	0.000048	0.000018	9.849921e-07	0.999965
1	0.000048	0.000047	9.993369e-01	0.000393
2	0.000022	0.000442	9.995970e-01	0.000079
3	0.287869	0.008124	8.851427e-05	0.740902
4	0.000012	0.000028	5.743566e-05	0.999894
...	...	...	...	...
1276	0.940327	0.034539	6.961226e-05	0.029514
1277	0.000141	0.000007	9.999634e-01	0.000017
1278	0.000606	0.000059	8.090886e-01	0.340588
1279	0.011510	0.000378	6.929170e-01	0.438285
1280	0.000052	0.028207	9.941655e-01	0.000590

1281 rows × 4 columns

In [ ]:

```
predicted_test_labels.columns=["hate","offen","prof","none"]
```

In [ ]:

```
pred_labels=[]
for i in range(predicted_test_labels.shape[0]):
  temp =
max(predicted_test_labels['hate'].iloc[i],predicted_test_labels['offen'].iloc[
i],predicted_test_labels['none'].iloc[i],predicted_test_labels['prof'].iloc[i])
  if temp ==
    predicted_test_labels['hate'].iloc[i]:
    pred_labels.append('HATE')
  elif temp ==
    predicted_test_labels['offen'].iloc[i]:
    pred_labels.append('OFFN')
  elif temp ==
    predicted_test_labels['prof'].iloc[i]:
    pred_labels.append('PRFN')
  elif temp ==
    predicted_test_labels['none'].iloc[i]:
    pred_labels.append('NONE')
```

In [ ]:

```
true_labels=test_mul_labels.label
```

In [ ]:

```
from sklearn.metrics import classification_report
```

In [ ]:

```
true_labels=list(true_label
s)
```

performance on hasoc test data

In [ ]:

```
print(classification_report(true_labels,pred_labels))
```

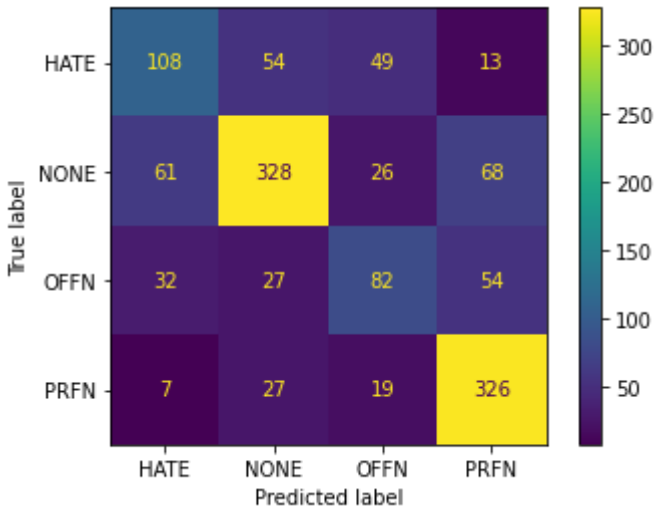
```
              precision    recall  f1-score   support

        HATE       0.52      0.48      0.50       224
        NONE       0.75      0.68      0.71       483
        OFFN       0.47      0.42      0.44       195
        PRFN       0.71      0.86      0.78       379

    accuracy                           0.66      1281
   macro avg       0.61      0.61      0.61      1281
weighted avg       0.65      0.66      0.65      1281
```

In [ ]:

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

In [ ]:

```
cm = confusion_matrix(true_labels, pred_labels,
labels=["HATE","NONE","OFFN","PRFN"]) disp =
ConfusionMatrixDisplay(confusion_matrix=cm,
                       display_labels=["HATE","NONE","OFFN","PRFN"])
disp.plo
t()
plt.show
()
```




```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       240
           1       0.69      1.00      0.82       529

    accuracy                           0.69       769
   macro avg       0.34      0.50      0.41       769
weighted avg       0.47      0.69      0.56       769
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       483
```

In [ ]:

```
true_labels=list(true_labels)
```

```
           1       0.62      1.00      0.77       798

    accuracy                           0.62      1281
   macro avg       0.31      0.50      0.38      1281
weighted avg       0.39      0.62      0.48      1281
              precision    recall  f1-score   support

           0       0.75      0.57      0.65       483
           1       0.77      0.89      0.82       798

    accuracy                           0.77      1281
   macro avg       0.76      0.73      0.74      1281
weighted avg       0.76      0.77      0.76      1281
              precision    recall  f1-score   support

           0       0.87      1.00      0.93      6216
           1       0.00      0.00      0.00       945

    accuracy                           0.87      7161
   macro avg       0.43      0.50      0.46      7161
weighted avg       0.75      0.87      0.81      7161
              precision    recall  f1-score   support

           0       0.93      0.98      0.95      6216
           1       0.79      0.53      0.64       945

    accuracy                           0.92      7161
   macro avg       0.86      0.75      0.80      7161
weighted avg       0.91      0.92      0.91      7161
              precision    recall  f1-score   support

           0       0.38      1.00      0.55       483
           1       0.00      0.00      0.00       798

    accuracy                           0.38      1281
   macro avg       0.19      0.50      0.27      1281
weighted avg       0.14      0.38      0.21      1281
              precision    recall  f1-score   support

           0       0.59      0.75      0.66       483
           1       0.82      0.68      0.74       798

    accuracy                           0.71      1281
   macro avg       0.70      0.72      0.70      1281
weighted avg       0.73      0.71      0.71      1281
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       141
           1       0.31      1.00      0.48       240
           2       0.00      0.00      0.00       140
           3       0.00      0.00      0.00       248

    accuracy                           0.31       769
   macro avg       0.08      0.25      0.12       769
weighted avg       0.10      0.31      0.15       769
              precision    recall  f1-score   support

           0       0.50      0.49      0.49       141
           1       0.62      0.70      0.66       240
           2       0.57      0.26      0.35       140
```

In [ ]:

```
true_labels=list(true_label
s)
```

```
              precision    recall  f1-score   support

           0       0.49      0.44      0.47       224
           1       0.66      0.69      0.68       483
           2       0.48      0.22      0.30       195
           3       0.69      0.89      0.78       379

    accuracy                           0.63      1281
   macro avg       0.58      0.56      0.55      1281
weighted avg       0.61      0.63      0.61      1281
```

```
              precision    recall  f1-score   support

           3       0.73      0.88      0.80       248

    accuracy                           0.64       769
   macro avg       0.61      0.58      0.58       769
weighted avg       0.63      0.64      0.62       769
```