SE 4050 – Deep Learning

4th Year second semester

Lab Assignment 1

Lasal Sandeepa Hettiarachchi

IT19132310

B.Sc. (Hons) in Information Technology Specializing in Software Engineering

Submitted to

Department of Computer Science and Software Engineering

Sri Lanka Institute of Information Technology

Sri Lanka

TABLE OF CONTENTS

TABLE OF FIGURES

1. PROBLEM DEFINITION

Technology has brought us closer than ever, and information is available at your fingertip in a matter of seconds. One of the best upsides to this technological advancement is the right to speech, meaning that we can express our opinions freely. But one of the greatest threats to freedom of speech is the difference in views and the harassment that people go through in an online environment. This is quite common in social media platforms such as Twitter, YouTube and Facebook, where there are millions of users. The Internet's ability to share knowledge is built on platforms that collect user material. blogs, forums, and message boards. The problem is that not everyone on the Internet wants to interact civilly; others may use it as a platform to express their anger, insecurities, and prejudices.

Cyberbullying or social media harassment can take many forms, especially when it comes to social media platforms. Some of the most common forms of social media harassment are trolling, stalking, disgracing, curse words, and racial hate speech. In this problem, we try to take an NLP-based approach to tackle this issue by creating a filtering layer which can identify toxic comments, obscene comments, threats, insults, and identity hate. This can be used to filter the comments or descriptions that are posted on social media platforms. This can help improve the quality of life on those platforms. The platform on that the dataset is based is Wikipedia comments. But the model can be applied anywhere.

2. DATASET DESCRIPTION

The Kaggle problem Toxic comment classification challenge is the baseline for this problem, and all the datasets used corresponds to that problem itself. This is part of a large number of datasets made publicly available by the Conversational AI team owned by alphabet to develop a large open dataset of Wikipedia comments. The dataset contains several subcategories to represent different areas of toxicity.

The train dataset contains, comments and the labels for subcategory of toxicity,

- Toxic
- Severe_toxic
- Obscene
- Threat
- Insult
- Indentity_hate

Along with the respective comments.

Resource:

https://www.kaggle.com/competitions/jigsaw-toxic-comment-classification-challenge/data

Kaggle download:

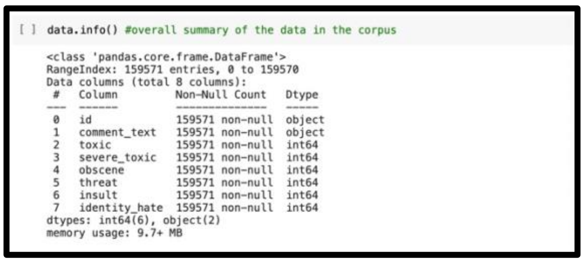
kaggle competitions download -c jigsaw-toxic-comment-classification-challenge.

The training dataset contains 159,570 entries whilw the testing dataset contains 153,531 entries.

The test.csv contains the training dataset along with their binary labels while the test.csv iis the test set . test_lebels.csv includes the test data of value -1. All the datasets are in the Comma separated value format(CSV). In the following section The exploratory data analysis part is described

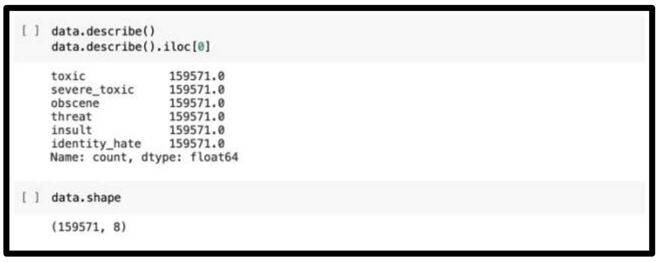
3. EXPLORATORY DATA ANALYSIS(EDA)

In this section, the exploratory data analysis pertaining to the dataset will be performed. Firstly the general information will be displayed after the **corpus** is imported to the dataframe from the CSV file



```
[ ] data.info() #overall summary of the data in the corpus

    <class 'pandas.core.frame.DataFrame'>
    RangeIndex: 159571 entries, 0 to 159570
    Data columns (total 8 columns):
     #   Column         Non-Null Count   Dtype
    ---  ------         --------------   -----
     0   id             159571 non-null  object
     1   comment_text   159571 non-null  object
     2   toxic          159571 non-null  int64
     3   severe_toxic   159571 non-null  int64
     4   obscene        159571 non-null  int64
     5   threat         159571 non-null  int64
     6   insult         159571 non-null  int64
     7   identity_hate  159571 non-null  int64
    dtypes: int64(6), object(2)
    memory usage: 9.7+ MB
```

Figure 2:Dataset summery

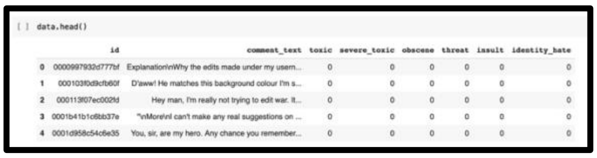


```
[ ] data.describe()
    data.describe().iloc[0]

    toxic            159571.0
    severe_toxic     159571.0
    obscene          159571.0
    threat           159571.0
    insult           159571.0
    identity_hate    159571.0
    Name: count, dtype: float64

[ ] data.shape

    (159571, 8)
```

Figure 1:Dataset summery

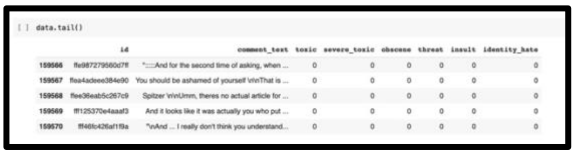
There are 8 columns, each containing 159,751 entries. The first column is an id column, while the second column is the commented text containing the toxicity. The labels toxic , severe_toxic, obscene , threat, insult and identity hate is of type int. The entries does not contain any null values.

The shape of the dataframe os 159,571 x 8



```
[ ] data.head()

                 id                                        comment_text  toxic  severe_toxic  obscene  threat  insult  identity_hate
   0  0000997932d777bf  Explanation\nWhy the edits made under my usern...      0             0        0       0       0              0
   1  000103f0d9cfb60f  D'aww! He matches this background colour I'm s...      0             0        0       0       0              0
   2  000113f07ec002fd      Hey man, I'm really not trying to edit war. It...      0             0        0       0       0              0
   3  0001b41b1c6bb37e   "\nMore\nI can't make any real suggestions on ...      0             0        0       0       0              0
   4  0001d958c54c6e35  You, sir, are my hero. Any chance you remember...      0             0        0       0       0              0
```

Figure 4:Dataset head

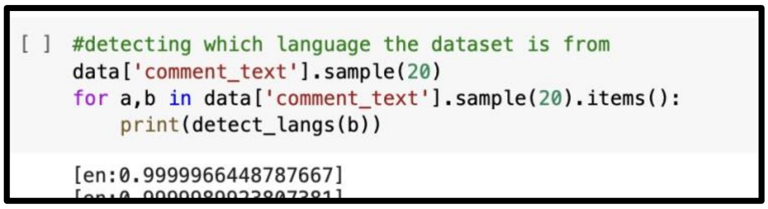


```
[ ] data.tail()

                      id                                        comment_text  toxic  severe_toxic  obscene  threat  insult  identity_hate
   159566  ffe987279560d7ff     ":::::And for the second time of asking, when ...      0             0        0       0       0              0
   159567  ffea4adeee384e90  You should be ashamed of yourself \n\nThat is ...      0             0        0       0       0              0
   159568  ffee36eab5c267c9    Spitzer \n\nUmm, theres no actual article for ...      0             0        0       0       0              0
   159569  fff125370e4aaaf3     And it looks like it was actually you who put ...      0             0        0       0       0              0
   159570  fff46fc426af1f9a    "\nAnd ... I really don't think you understand...      0             0        0       0       0              0
```

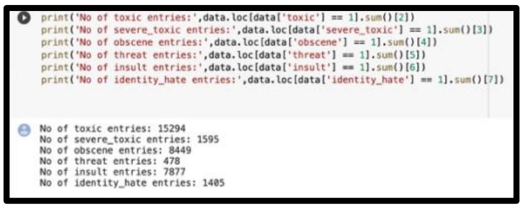
Figure 3:Dataset tail

The head and the tail of the data frame are as follows. We can observe that none of the entries above does have any form of toxicity. To be more specific, only around 10% of the entries contain any form of toxicity. We can also observe a lot of punctuation and capitalizations, which have to be addressed under the data preparation.

As a preprocessing technique, the stop words and other words associated with language need to be removed therefore firstly the language of the dataset has to be identified. Also, The number of each counted label are displayed below. Accordingly there are 15294 toxic comments,1595 severe toxic comments, 8449 obscene comments, 478 threats, 7877 insults and 1405 identity hate comments



```
[ ]  #detecting which language the dataset is from
     data['comment_text'].sample(20)
     for a,b in data['comment_text'].sample(20).items():
         print(detect_langs(b))

     [en:0.9999966448787667]
```



```
print('No of toxic entries:',data.loc[data['toxic'] == 1].sum()[2])
print('No of severe_toxic entries:',data.loc[data['severe_toxic'] == 1].sum()[3])
print('No of obscene entries:',data.loc[data['obscene'] == 1].sum()[4])
print('No of threat entries:',data.loc[data['threat'] == 1].sum()[5])
print('No of insult entries:',data.loc[data['insult'] == 1].sum()[6])
print('No of identity_hate entries:',data.loc[data['identity_hate'] == 1].sum()[7])

No of toxic entries: 15294
No of severe_toxic entries: 1595
No of obscene entries: 8449
No of threat entries: 478
No of insult entries: 7877
No of identity_hate entries: 1405
```

The heat map between the correlations of each label can be identified below. From this, it can be observed that there is a high correlation between obscene comments and an insult. There is also a high chance that a toxic comment is obscene as well as an insult

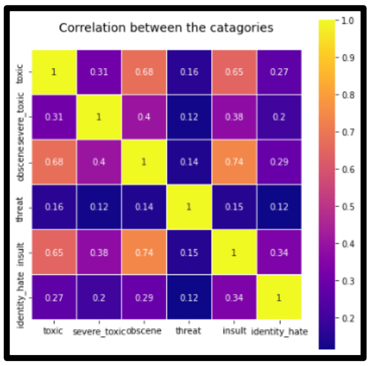
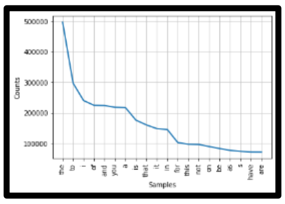
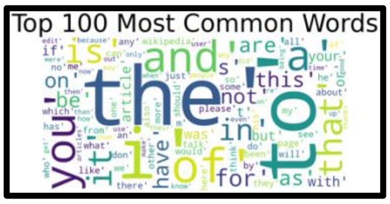


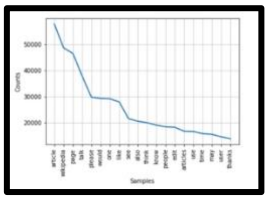
Figure 5:Heatmap between the labels

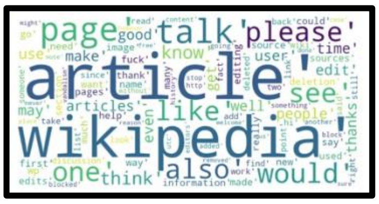
The motive is to tokenize each word in the vocabulary of the corpus and write each word in the document with encoding. But before doing that we first have to identify that some words don't hold any meaning in the context of understanding the toxicity. Also, the root word of a word might be enough to identify the importance of the word. Therefore, lemmatization and removal of stop words punctuation and other tricks are performed.

If the word cloud of the corpus is observed, we can see that the majority of the words are stop words which have no importance to the word meaning but influences the model greatly. Therefore, they have to be removed.







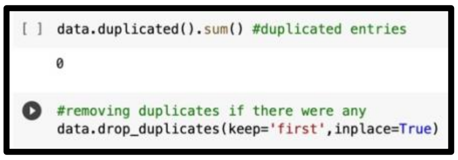


It can be observed how the word frequencies change after stop word removal. Before the removal of the stop words, the occurrence of the words [('the', 497217), ('to', 297521),('of', 225084),]. But after stop word removal, the highest occurring words were. [('article', 57738), ('wikipedia', 48625), ('page', 46542), ('talk', 37908)].

4. DATA PREPROCESSING

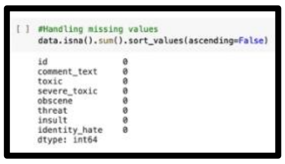
In the lexical analysis and NLP problems, the data preprocessing part is as equally as important as the model architecture process. The accuracy and the efficiency of training and the prediction process solely relies on how the preprocessing part Is handled. Under this section the following tasks are performed

- Looking for entries with null values and if present manually catagorize the entries if the number of entries with null entries is small.
- Looking for duplicated entries and removing them if they are present
- Removing all punctuation and numbers
- Lowercasing the words since the lowercase and uppercase words doesnt hold any value in detecting toxicity
- From the above graph it can be observed that the heightest frequency of the words that are occouring are stop words . Since they does not hold any value to the context of our problem, we will remove them
- Rephrasing contractions
- Lematizing the words to their roots since we can derive the meaning from them yet with a lesser number of values to encode.

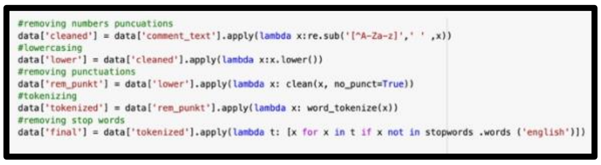


```
[ ] data.duplicated().sum() #duplicated entries

    0

    #removing duplicates if there were any
    data.drop_duplicates(keep='first',inplace=True)
```

Figure 7:Removing duplicate entries



```
[ ] #Handling missing values
    data.isna().sum().sort_values(ascending=False)

    id               0
    comment_text     0
    toxic            0
    severe_toxic     0
    obscene          0
    threat           0
    insult           0
    identity_hate    0
    dtype: int64
```

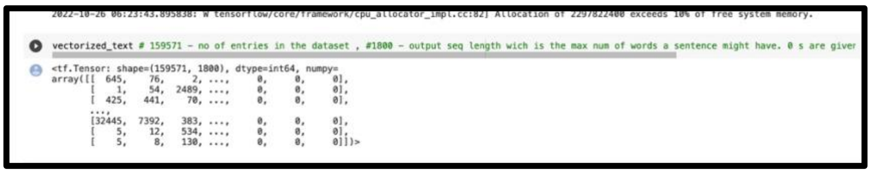


```
#removing numbers puncuations
data['cleaned'] = data['comment_text'].apply(lambda x:re.sub('[^A-Za-z]',' ' ,x))
#lowercasing
data['lower'] = data['cleaned'].apply(lambda x:x.lower())
#removing punctuations
data['rem_punkt'] = data['lower'].apply(lambda x: clean(x, no_punct=True))
#tokenizing
data['tokenized'] = data['rem_punkt'].apply(lambda x: word_tokenize(x))
#removing stop words
data['final'] = data['tokenized'].apply(lambda t: [x for x in t if x not in stopwords .words ('english')])
```

Figure 6:Data preprocessing process

The text vectorization process is one of the most important processes in NLP, the vectorization encodes all the vocabulary in the corpus and creates vectors for each document with the encoding corresponding to each word. If the word is not present in the sentence, all other values will be 0.Lametization: In linguistics, this is the process of combining a word's inflected forms into a single unit for analysis using the word's lemma, or dictionary form.

The input dataset is then separated into batches of 16 to be trained batch-wise to improve performance and overcome bottlenecks. The following displays an encoded batch of such documents.



```
2022-10-26 06:23:43.895838: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 2297822400 exceeds 10% of free system memory.

vectorized_text # 159571 - no of entries in the dataset , #1800 - output seq length wich is the max num of words a sentence might have. 0 s are giver

<tf.Tensor: shape=(159571, 1800), dtype=int64, numpy=
array([[  645,    76,     2, ...,     0,     0,     0],
       [    1,    54,  2489, ...,     0,     0,     0],
       [  425,   441,    70, ...,     0,     0,     0],
       ...,
       [32445,  7392,   383, ...,     0,     0,     0],
       [    5,    12,   534, ...,     0,     0,     0],
       [    5,     8,   130, ...,     0,     0,     0]])>
```

Figure 8:Encoded batch of documents

The train dataset is then split into 3 datasets as trainDataset, validationDataset and testDataset in the ratio of [7:2:1]

5. LEARNING ALGORITHM

The problem at hand is essentially a text classification problem to label whether a comment or a piece of text falls into toxic comments, obscene comments, threats, insults, and identity hate categories. Although at first this seems like a multi class classification at first since there are multiple labels, this is essentially a binary classification applied multiple times. To be more specific this is a one to one binary classification applied multiple times . Since texts are of variable length, maintain long sequences, retains information in order and the meaning changes with order, shares parameters across sequences, the deep learning approach that is recommended should be is a recurrent neural network model that can perform all the aforementioned values. To be more specific, a flavour of an RNN called LSTM was used.

5.1 RNN

RNNs are a type of neural network which excel at processing sequential data such as sentences. These can memorize input patterns of any duration.RNNs, however, are unable to identify long-term relationships in prolonged sequences because of the failure of stochastic gradients. Therefore, a special kind of RNN, called BiLSTM, which addresses the stochastic gradient problem, is used.

5.2 BiLSTM

LSTM directly addresses the vanishing gradient problem in RNNs through gates. This reduces the loss of information. The forget gate, update gate, store gate and output gate helps in the process. The advantage of using the BiLSTM in this scenario is that it allows us to pass information forward and backwards across layers. This is particularly important in sentences since words prior to or after a word may modify the meaning.

The layers of the BiLSTM architecture are as follows. Firstly, as text vectorization layer mentioned under the data preprocessing part is used. The vectorization layer is directly connected to embedding layer which embeds all the words (Max_words) to a vector of 32 per each word. Embedding layer keeps the qualities about the personality of each word and helps identify which words may lead to toxic comments. The embedding layer is then connected to an LSTM with a bidirectional modifier with tanh activation function. The LSTM is then connected to 3 consequent Dense layes with 128,256 and 128 units each with activation function of ReLU.These are primarily there for the feature extraction process. The final layer is a dense layer with the number of units equivalent to the output labels.

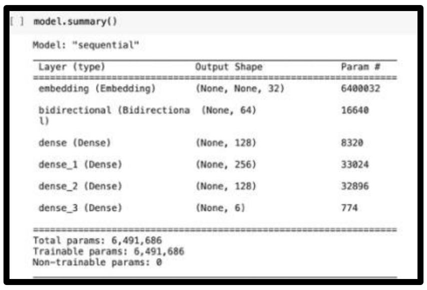
The following is a discussion of why each activation function and loss function are used

Sigmoid : to output a value between 0-1 and to add linearity to the model. This is used in the output layer since the output for each label should either be 0 or 1

Tanh : regularization of the output from the LSTM layer.

Binerrycrossentrpy: This is used as the loss function since it is required for us to reduce the loss of the binary labels predicted. As explained in the earlier section. Although this may seem like a multiclass Classification, this is essentially a binary classification per each label

Optimizer : Adam



```
[ ] model.summary()

    Model: "sequential"
    _________________________________________________________________
     Layer (type)                Output Shape              Param #
    =================================================================
     embedding (Embedding)       (None, None, 32)          6400032

     bidirectional (Bidirectiona  (None, 64)               16640
     l)

     dense (Dense)               (None, 128)               8320

     dense_1 (Dense)             (None, 256)               33024

     dense_2 (Dense)             (None, 128)               32896

     dense_3 (Dense)             (None, 6)                 774

    =================================================================
    Total params: 6,491,686
    Trainable params: 6,491,686
    Non-trainable params: 0
    _________________________________________________________________
```

Figure 10:Model summery

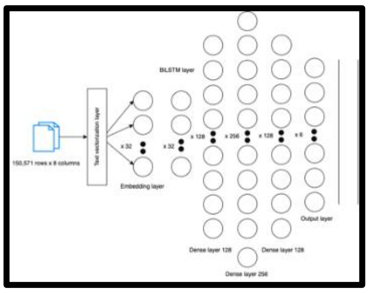


Figure 9:BiLSTM architecture

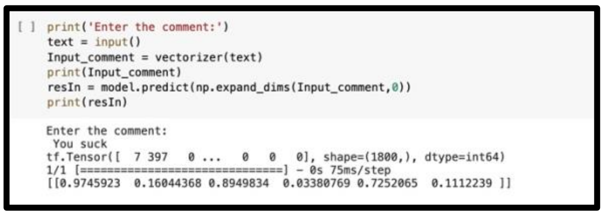
As further improvements, dropout layers can be added to reduce the overfitting problem.

Input to the model: The output from the text vectorization layer which is a 159,571 x 1800 vector. Divided into batches of 16

Loss function: In binary cross entropy the actual class output, which can only be either 0 or 1, is compared to each of the projected probabilities. The score that penalizes the probabilities based on how far they are from the predicted value is then calculated. Depending on how near or far the value is to the actual value.

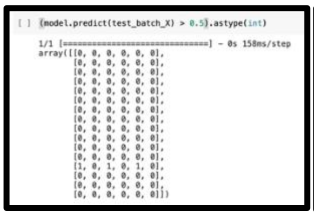
6. TEST RESULTS AND ANALYSIS

The model's prediction process returns a float value array containing all the values then a condition can be used to give a binary output for each label. This method was used iteratively for all the batches in the test dataset and an output for a single batch of prediction and the actual value is shown below



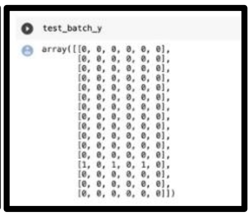
```
[ ] print('Enter the comment:')
    text = input()
    Input_comment = vectorizer(text)
    print(Input_comment)
    resIn = model.predict(np.expand_dims(Input_comment,0))
    print(resIn)

    Enter the comment:
     You suck
    tf.Tensor([  7 397   0 ...   0   0   0], shape=(1800,), dtype=int64)
    1/1 [==============================] - 0s 75ms/step
    [[0.9745923  0.16044368 0.8949834  0.03380769 0.7252065  0.1112239 ]]
```

Figure 11:Prediction per single input



```
[ ] (model.predict(test_batch_X) > 0.5).astype(int)

    1/1 [==============================] - 0s 158ms/step
    array([[0, 0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0, 0],
           [1, 0, 1, 0, 1, 0],
           [0, 0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0, 0]])
```

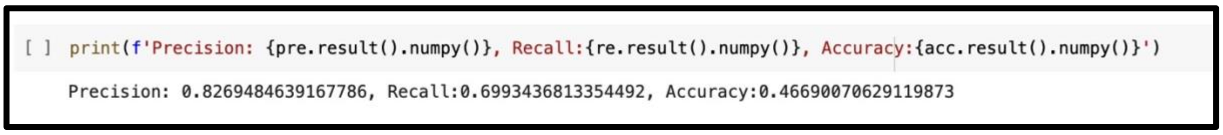
Figure 13:Actual prediction



```
test_batch_y

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

Figure 12:Actual value

When comparing the 3 metrics used for evaluating the model, it can be identified that recall is not a good metric to evaluate the model. Recall is the ratio of positive predictions to all samples with that label, and specificity is the ratio of negative predictions to all negative samples. But As explored earlier, less than 10% of the entries trigger a positive value. Recall would've been a good metric if the relative distribution of positive and negative labels were even. Rather precision is the ratio between the number of true positives and the total number of true positives. It is also possible to use the F1 score for this purpose since it is the harmonic average of between precision and recall, it enhances their strengths while subsidizing the weaknesses.



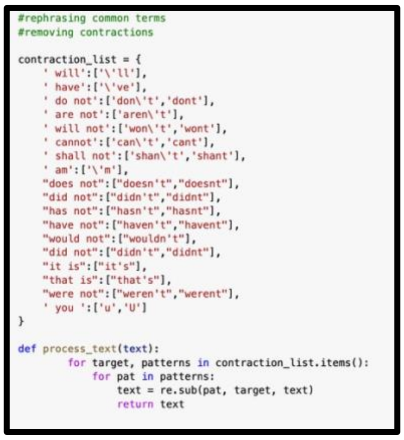
```
[ ] print(f'Precision: {pre.result().numpy()}, Recall:{re.result().numpy()}, Accuracy:{acc.result().numpy()}')

    Precision: 0.8269484639167786, Recall:0.6993436813354492, Accuracy:0.46690070629119873
```

7. FURTHER IMPROVEMENTS

It is possible to observe that the model was run for only a single epoch with hardware limitations. But the training validation loss was still decreasing. Therefore as further improvements in order to increase the accuracy of the model, the training process can be run for an increasing number of epochs without overfitting the model.

Also, the current trend is evolving from BiLSTM and GRUs to transformer networks for NLP tasks, this is because they yield a higher accuracy and efficient at making predictions than RNNs. There are existing open source tools such as BERT as well (bidirectional encoder representations from transformers).

Also, it can be observed that words may appear in several ways and forms. Further, hate speakers and toxic person may try to hide the hate speech by typing it in uncommon ways. This cannot be identified by lemmatization. For an example the word retard can be identified in many forms in the texts {'retard' : ['returd', 'retad', 'retard', 'wiktard', 'wikitud']} similarly there are many other words. Also abbreviations such as 'stfu' may be used. Also people may repeat a letter multiple times to avoid detections. Such as in 'iiiiiidiott' => 'idiot'. These needs to address by the preprocessing phase. Although contractions were removed in the above approach, the instances that are discussed must be addressed separately.



```
#rephrasing common terms
#removing contractions

contraction_list = {
    ' will':['\'ll'],
    ' have':['\'ve'],
    ' do not':['don\'t','dont'],
    ' are not':['aren\'t'],
    ' will not':['won\'t','wont'],
    ' cannot':['can\'t','cant'],
    ' shall not':['shan\'t','shant'],
    ' am':['\'m'],
    "does not":["doesn't","doesnt"],
    "did not":["didn't","didnt"],
    "has not":["hasn't","hasnt"],
    "have not":["haven't","havent"],
    "would not":["wouldn't"],
    "did not":["didn't","didnt"],
    "it is":["it's"],
    "that is":["that's"],
    "were not":["weren't","werent"],
    ' you ':['u','U']
}

def process_text(text):
        for target, patterns in contraction_list.items():
            for pat in patterns:
                text = re.sub(pat, target, text)
                return text
```

Figure 14:Contraction list

Also, while the combination of binary cross entropy and ADAM was used as the optimizer and the loss, there might be other combinations that may lead to higher accuracy.

Also, in order to increase accuracy methods such as ensemble learning can be used combining several models to give a high accuracy prediction. This might be a combination of Deep learning and machine learning based models.