
# Final Project: NLP – Twitter Hate Speech Detection Using Transformers (Deep Learning)

Student Name: Manoj Kumar Thangaraj

E-mail: manojthangaraj92@gmail.com

Module Title: Data & Web Mining

Module Code: LISUM01

Specialization: Natural Language Processing

# Contents

## 1. Project overview/scope

The main aim of this project is to develop a hate speech detection model using Transformers architecture in deep learning. The term hate speech is understood as any type of verbal, written or behavioral communication that attacks or uses derogatory or discriminatory language against a person, or group based on what they are, in other words, based on their religion, ethnicity, nationality, race, colour, ancestry, sex, or any other identity factor.

It is basically a form of Classification task. Therefore, for training, a model that can classify hate speech from a certain piece of text can be achieved by training it on a data that is generally used to classify sentiments.

## 2. Data Source

The data source is given by the company. The dataset is available in the following https://www.kaggle.com/vkrahul/twitter-hate-speech?select=train_E6oV3lV.csv. We express out gratitude to Mr. Rahul Agarwal for this dataset. The dataset is downloaded on the Jupyter Notebook using the open-source library OpenDatasets.

```
In [97]: import opendatasets as od
         train_tweets = od.download("https://www.kaggle.com/vkrahul/twitter-hate-speech?select=train_E6oV3lV.csv")

         Skipping, found downloaded files in "./twitter-hate-speech" (use force=True to force download)
```

## 3. Preparing and Pre-processing data

The important step of this whole project is pre-processing the data. This is very important in terms of replacing the words, symbols, HTML tags that are not meaningful in anyway. These items present in the data will lead to false predictions, directing the training in a wrong way.

The library BeautifulSoup and Regex has been used to take out the words that are not meaningful. The function review_to_words is defined to do this job. This function will return the processed text back to us.

In addition to that, function read_csv is defined to take the above function and do some additional steps before we can get the full data. The snippet of the code is presented down below.

In [189]:
```
#import the required libraries
import pandas as pd
import numpy as np
import re
from bs4 import BeautifulSoup

def review_to_words(review):
    '''Function that takes in the word and remove html parser and other symbols.
    Args: review, the sentence in the string format
    returns: cleansed text'''
    text = BeautifulSoup(review, "html.parser").get_text() # Remove HTML tags
    text = re.sub(r"[^a-zA-Z0-9]", " ", text.lower()) # Convert to lower case
    return text

def read_csv(file_path):
    '''Function that takes in the file and returns in the format cleaned and ready for feature learning
    Args: csv file
    returns: datarfame'''
    df = pd.read_csv(file_path)          #read in the file
    df = df.drop(['id'], axis=1)         #drop the unecessary columns
    df['tweet'] = df['tweet'].apply(lambda x:review_to_words(x))   #apply the above funtion to preporcess the word
    df = df[['tweet','label']]           #rearrange the columns
    return df       #returns dataframe

#Now, specify the path that file located
path = r'/home/ec2-user/SageMaker/twitter-hate-speech/train_E6oV3lV.csv'

#apply the function and view the dataframe
df = read_csv(path)
df.head()
```

The snippet of the output is given below.

Out[189]:

| | tweet | label |
|---|---|---|
| 0 | user when a father is dysfunctional and is s... | 0 |
| 1 | user user thanks for lyft credit i can t us... | 0 |
| 2 | bihday your majesty | 0 |
| 3 | model i love u take with u all the time in ... | 0 |
| 4 | factsguide society now motivation | 0 |

Once the above process is finished and we got our results, a new dictionary has been developed for locally mistyped words and their correct words as replacements. Lambda function is developed to send these keys, value pairs to look for keys that are present in the dataset and to replace them with their values in the dataset. Developing this dictionary will eventually improve the results.

In [190]:
```
#Code for removing mistake words
replace_words = {'luv':'love','wud':'would','lyk':'like','wateva':'whatever','ttyl':'talk to you later',
                 'kul':'cool','fyn':'fine','omg':'oh my god!','fam':'family','bruh':'brother',
                 'cud':'could','fud':'food'} ## Need a huge dictionary

#we will pass a sentence to test our function
sentence = "I luv myself"
words = sentence.split()    #split the sentence into words

#replace the words in the sentence with the word in the dictionary if the the actual word itself.
reformed = [replace_words[word] if word in replace_words else word for word in words]
reformed = " ".join(reformed)    #join the splitted after reforming
print(reformed)     #print the reformed
```

```
I love myself
```

The above function worked and the word 'luv' ir replaced by 'love' from the dictionary. As I said, improving this dictionary will help us improve the end results.

**Now let us apply this dictionary in our table and get it reformed.**

In [191]:
```
#apply the function on df['tweet']
df['tweet'] = df['tweet'].apply(lambda x : ' '.join(replace_words[word] if word in replace_words else word for word in x.split()))

#disply the reformed
df.head()
```

Out[191]:

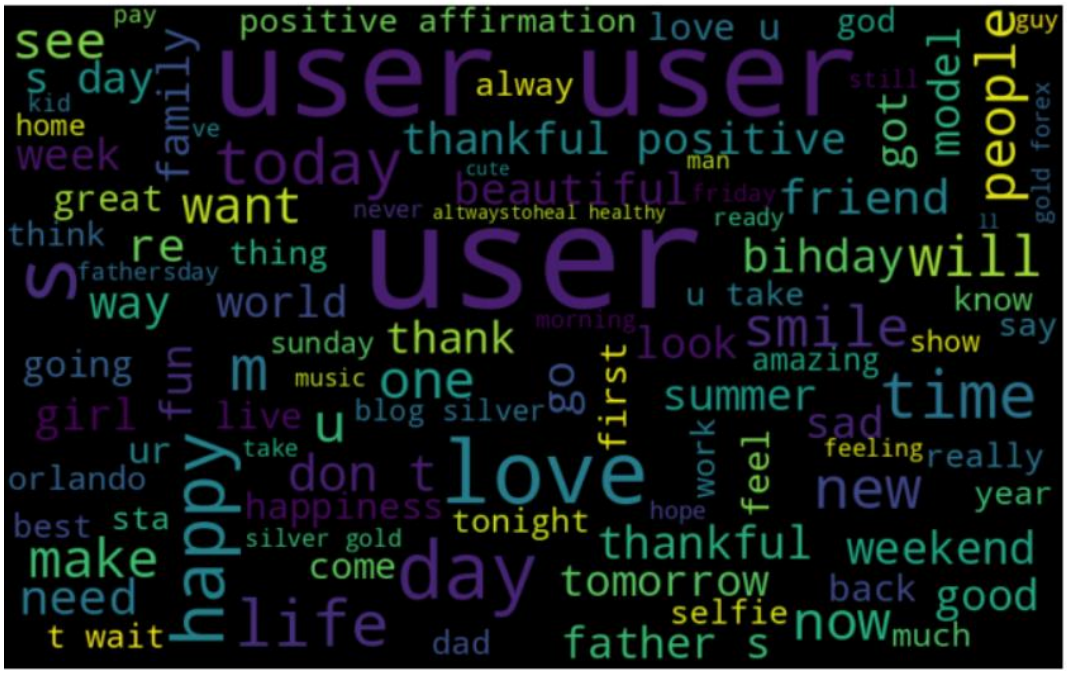
| | tweet | label |
|---|---|---|
| 0 | user when a father is dysfunctional and is so ... | 0 |
| 1 | user user thanks for lyft credit i can t use c... | 0 |
| 2 | bihday your majesty | 0 |
| 3 | model i love u take with u all the time in ur | 0 |
| 4 | factsguide society now motivation | 0 |

## 4. Data Visualization

Though it's the textual data, there is always way to visualize data. In this part we picked up the wordcloud library to show the usage of the words. Eventually, it helped us to go back to the processing part and do some further processing to the good quality data.

Also, the balance level between the labels has been checked and found there is high level of imbalance found between the labels. This is something to be addressed.

In [5]:
```
#Import the libraries
from wordcloud import WordCloud
import matplotlib.pyplot as plt

#take all words in df['tweet'] with the labels 0 for visualization
normal_words = ' '.join([word for word in df['tweet'][df['label'] == 0]])
#set the wordcloud parameters
wordcloud = WordCloud(width = 800, height = 500, max_font_size = 110,max_words = 100).generate(normal_words)
print('Normal words')
#plot the graph
plt.figure(figsize= (12,8))
plt.imshow(wordcloud, interpolation = 'bilinear',cmap='viridis')
plt.axis('off')
```




Upon visualizing the picture, we found that the word 'user' has been used many times. This is adding no meaning to text and needs replacement. It can be added to dictionary with empty space as value which is not an ideal way to do it. Therefore, we defined a lambda function to replace this word with empty space.

```
#replace the word user with empty space
df['tweet'] = df['tweet'].apply(lambda x: x.replace('user',''))

#print the dataframe
df.head()
```

| | tweet | label |
|---|---|---|
| 0 | when a father is dysfunctional and is so self... | 0 |
| 1 | thanks for lyft credit i can t use cause the... | 0 |
| 2 | bihday your majesty | 0 |
| 3 | model i love u take with u all the time in ur | 0 |
| 4 | factsguide society now motivation | 0 |

There is high Imbalance between the label 0 and 1. To tackle this problem, we have many ways to do it. There is two popular ways to this is *oversampling** by increase the number tweets with label 1. We can do this with the help of SMOTE overspampling technique. But if we can see, it's approximately 12 times higher than label 1. In our case, upsampling might lead to overfitting in the training.

The another is undersampling, which is good for this type of problem. The disadvantage in picking them up is we are going to loss so much information in training this model. But upon development, and collection of data in the future and updating the model, this problem can be resolved.

Therefore, we are picking undersampling technique for this problem and continue our model development and training process.

**Underspampling technique**

```
#divide the dataframe with each class labels
df_positive = df[df['label']==0]
df_negative = df[df['label']==1]

#sample only 2500 out of all from the label 0
df_positive = df_positive.sample(2500)

#append the df with label 1 with label 0
df_positive  = df_positive.append(df_negative)

#Shuffle the df
from sklearn.utils import shuffle
df = shuffle(df_positive)

#reset the index
df = df.reset_index(drop=True)
df.head()
```

| | tweet | label |
|---|---|---|
| 0 | cont day 3 of 4 d dl2016 downloadfestival meta... | 0 |
| 1 | aymtracklist no seasoned aist vocals kahik chi... | 0 |
| 2 | mcconnell obstruction goes all the way back t... | 1 |
| 3 | stop teasing us with these pics i m not slee... | 0 |
| 4 | the reason is said to be mr perfectionist is ... | 0 |

## 5. Transforming the data

Deep learning models generally prefer their training data to be represented by numbers i.e., tensors in arrays or NumPy arrays. For this requirement, the text formats in our dataset needs to be transformed into tensors which our deep learning model can communicate to. In other words, tokenizing each word present in our dataset based their number occurrences in the corpus. Also, fixing the sequence length so that the model knows all the inputs comes in that range. Padding for the minimum occurrences and truncating the largest ones will help us do this.

Padding is a process of adding an extra token called padding token at the beginning or end of the sentence. As the number of the words in each sentence varies, we convert the variable length input sentences into sentences with the same length by adding padding tokens. Padding is required since most of the frameworks support static networks, i.e., the architecture remains the same throughout the model training.

Torch is the important package which is used tot transform to tensors and mathematical operations on it. TorchText is a Natural Language Processing (NLP) library in PyTorch. This library contains the scripts for pre-processing text and source of few popular NLP datasets.

We will define a function build_vocab, which has the following operations. Before that, there are two different types of fields, Objects-Field and Label-field. The former is the field object from the data module is used to specify pre-processing steps for each column in the dataset and the latter is the special case of field object which is only used for the classification tasks.

When we instantiate these fields first, there are parameters needs to be considered,

- Tokenize: specifies the way of tokenizing the sentence i.e., converting sentence to words. I am using spacy tokenizer since it uses novel tokenization algorithm
- Lower: converts text to lowercase
- batch_first: The first dimension of input and output is always batch size

Following the above steps, we will create a list of tuples in which every tuple contains object field and followed label field as defined above. In fact, we will arrange the tuples in accordance with our columns of the csv file. Since we have index as the first column, we will specify the tuple (None,None) to ignore that column.

Once we are done with instantiating the fields, we will load the pre-processed dataset in the torchtext tabular dataset function with the specified parameters. Now it's the time to split the dataset into train and validation dataset. We don't need test dataset since we have a separate test dataset. This is done with the help of random library with its split function which is excellent for splitting up the dataset.

The next step is to build the vocabulary for the text and convert them into integer sequences. Vocabulary contains the unique words in the entire text. Each unique word is assigned an index. Below are the parameters listed for the same

Parameters:

1. min_freq: Ignores the words in vocabulary which has frequency less than specified one and map it to unknown token.
2. Two special tokens known as unknown, and padding will be added to the vocabulary

- Unknown token is used to handle Out of Vocabulary words
- Padding token is used to make input sequences of same length

Let us build vocabulary and initialize the words with the pretrained embeddings. Ignore the vectors parameter if you wish to randomly initialize embeddings.

Now we will prepare batches for training the model. BucketIterator forms the batches in such a way that a minimum amount of padding is required. Once, the above process is completed, we will get returns train and test iterator objects, word dictionary and length of the vocabulary. The whole function snippet is given below.

```
import torch
from torch import nn
from torch.autograd import Variable
import torch.nn.functional as F
import random, tqdm, sys, math, gzip
from torchtext.legacy import data, datasets, vocab
import numpy as np
import random
import spacy


#defining the function to transform the data
def build_vocab(file_path):
    '''Function to take in the preprocessed file and transform into a iterators and word dictionaries
    Args: csv file
    returns: iterators, length of vocab, word_dict'''
    #Reproducing same results
    SEED = 2019

    #Torch
    torch.manual_seed(SEED)

    #Instantiate the fields
    TEXT = data.Field(tokenize='spacy',lower=True, include_lengths=True, batch_first=True)
    LABEL = data.LabelField(batch_first=True)
    #since the first column is the index, the tuple is left none
    fields = [(None,None),('tweet', TEXT),('label', LABEL)]

    #Load the file and build the torchtext dataset
    training_data=data.TabularDataset(path = file_path,format = 'csv',fields = fields,skip_header = True)

    #split the dataset into train and valid
    train_data, valid_data = training_data.split(split_ratio=0.7, random_state = random.seed(SEED))

    #build vocabulary
    TEXT.build_vocab(train_data,min_freq=3, vectors = "glove.6B.100d")
    LABEL.build_vocab(train_data)

    #check whether cuda is available
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

    #set batch size
    BATCH_SIZE = 64

    #Load an iterator
    train_iterator, valid_iterator = data.BucketIterator.splits(
        (train_data, valid_data),
        batch_size = BATCH_SIZE,
        sort_key = lambda x: len(x.tweet),
        sort_within_batch=True,
        device = device)
    len_text_vocab = len(TEXT.vocab)
    word_dict = TEXT.vocab.stoi
    #returns the required objects
    return train_iterator, valid_iterator, len_text_vocab, word_dict
```

We will provide the above function with path and in turn get the iterators and word dict.

```
#defining the path of the file
path = r'/home/ec2-user/SageMaker/train/df.csv'

#assigining it to a function to get iterators
train_it, test_it, len_text_vocab, word_dict = build_vocab(path)
```

```
/home/ec2-user/anaconda3/envs/pytorch_latest_p36/lib/python3.6/site-packages/torchtext/data/utils.py:123: UserWarning: Spacy model "en" c
ould not be loaded, trying "en_core_web_sm" instead
  warnings.warn(f'Spacy model "{language}" could not be loaded, trying "{OLD_MODEL_SHORTCUTS[language]}" instead')
```

```
#get the number of batches in each of the iterators

print(f'- nr. of training examples {len(train_it)}')
print(f'- nr. of testing examples {len(test_it)}')
```

```
- nr. of training examples 52
- nr. of testing examples 23
```

## 6. Model Building

For this project, we aim to build the transformers from scratch. i.e., without using any pre-trained models such BERT etc., Transformer are the very exciting family of machine learning architectures.

Transformers model is built with the help of PyTorch neural networks. For our model, we will build a self-attention layer which is embedded in transformer block which is then embedded in Classification Transformer Model. This is the high-level view of our transformer model.

We will go by explaining one by one. Firstly, the fundamental operation of any transformer architecture is the self-attention operation. It is a sequence-to-sequence operation where a sequence of vectors goes in, and a sequence of vectors comes out. Let's say vectors X1, X2, X3…. Xt, have their output vectors as Y1, Y2, Y3….Yt. The self-attention operation takes a weighted average over all the input vectors.

$$\mathbf{y}_i = \sum_j w_{ij} \mathbf{x}_j \ .$$

Where j indexes over the whole sequence and the weights sum to one over all j. The weight wij is not a parameter, as in a normal neural net, but it is *derived* from a function over $\mathbf{x}$i and $\mathbf{x}$j. The simplest option for this function is the dot product

$$w'_{ij} = \mathbf{x}_i^T \mathbf{x}_j \ .$$

The dot product gives us a value anywhere between negative and positive infinity, so we apply a SoftMax to map the values to [0,1] and to ensure that they sum to 1 over the whole sequence. This is the basic operation of self-attention.

But the dot product is not possible when there are not any features. In our case we have only text data. For this problem, we have a concept called key, value, query concept where the query is the input vector key and values are vectors that is matched against input vectors.

We will create this simple transformer as we go along. First the self-attention in pytorch. The first thing we should do is work out how to express the self attention in matrix multiplications. A naive implementation that loops over all vectors to compute the weights and outputs would be much too slow.

We'll represent the input, a sequence of t vectors of dimension k as a t by k matrix X. Including a minibatch dimension b, gives us an input tensor of size (b,t,k).
The set of all raw dot products w'ij forms a matrix, which we can compute simply by multiplying X by its transpose. Then to turn raw weights into the positive values that sum to one, we apply a row-wise SoftMax. Finally, to compute the output sequence, we just multiply the weight matrix by X. This results in a batch of output matrices Y of size (b, t,k) whose rows weights are sum over the rows of X. This two-matrix multiplication and one SoftMax gives us a basic self-attention.

Now we will build a transformer block, this transformer block will have a self-attention layer, a feed forward layer and another layer normalization. Residual connections are added around both, before the normalization. The important thing is to combine self-attention with a local feedforward, and to add normalization and residual connections.

Finally, now we will build the classification transformer. We will build a transformer blocks according to the number of depths. Before that whatever the tensors coming in, will have their own tensor value, along with their positions. For this we will have a positional embedding layer which will give a position to each word which is then added the embedding tensors. This will go through the dropout layer and the number of transformer blocks and finally to SoftMax value.

```
class CTransformer(nn.Module):
    """
    Transformer for classifying sequences
    """

    def __init__(self, emb, heads, depth, seq_length, num_tokens, num_classes, max_pool=True, dropout=0.0, wide=False):
        """
        emb: Embedding dimension
        heads: nr. of attention heads
        depth: Number of transformer blocks
        seq_length: Expected maximum sequence length
        num_tokens: Number of tokens (usually words) in the vocabulary
        num_classes: Number of classes.
        max_pool: If true, use global max pooling in the last layer. If false, use global
                  average pooling.
        """
        super().__init__()

        self.num_tokens, self.max_pool = num_tokens, max_pool

        self.token_embedding = nn.Embedding(embedding_dim=emb, num_embeddings=num_tokens)
        self.pos_embedding = nn.Embedding(embedding_dim=emb, num_embeddings=seq_length)

        tblocks = []
        for i in range(depth):
            tblocks.append(
                TransformerBlock(emb=emb, heads=heads, seq_length=seq_length, mask=False, dropout=dropout))

        self.tblocks = nn.Sequential(*tblocks)

        self.toprobs = nn.Linear(emb, num_classes)

        self.do = nn.Dropout(dropout)
```

```
    def forward(self, x):
        """
        x: A batch by sequence length integer tensor of token indices.
        return: predicted log-probability vectors for each token based on the preceding tokens.
        """
        tokens = self.token_embedding(x)
        b, t, e = tokens.size()

        positions = self.pos_embedding(torch.arange(t, device=device))[None, :, :].expand(b, t, e)

        x = tokens + positions
        x = self.do(x)
        x = self.tblocks(x)
        x = x.max(dim=1)[0] if self.max_pool else x.mean(dim=1) # pool over the time dimension
        x = self.toprobs(x)

        return F.log_softmax(x, dim=1)
```

## 7. Training and Validation

Training the model and validating the model is done using same loop. The classification loss and training validation is calculated during training and the test validation accuracy is calculated during the validation. Also, the original labels, predicted labels obtained to do evaluation and metrics.

The training and validation yielded a good result. For the 20 epochs, the training validation came up to 88.79% and test validation came up to 80.4%. The model is then saved for model inference.

```
classification/train-loss 0.17044620215892792 65877
classification/train-loss 0.12125111371278763 65941
classification/train-loss 0.06628582626581192 66005
 92%|████████▊ | 48/52 [00:04<00:00,  9.34it/s]
classification/train-loss 0.04061279818415642 66069
classification/train-loss 0.2805091440677643 66133
classification/train-loss 0.13782714307308197 66197
 98%|█████████▊| 51/52 [00:05<00:00,  8.67it/s]
classification/train-loss 0.1384420245885849 66252
classification/train-loss 0.17706452310085297 66316
100%|██████████| 52/52 [00:05<00:00,  9.88it/s]
classification/train-loss 0.263893187046051 66380
-- training validation accuracy 88.79481771617958
100%|██████████| 23/23 [00:00<00:00, 32.43it/s]
-- test validation accuracy 80.46380885453269
The model is saved
```

## 8. Evaluation and Metrics

From the training and validation process, the original labels and metrics are extracted. They are saved in a NumPy arrays. The general evaluation metrics such as confusion matrix, precision, recall and f1 scores are compared.

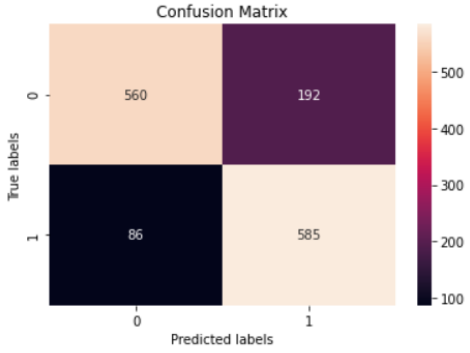
In terms of precision, which is true positives over the true positive and negatives, the label 0 achieved 87% and label 1 achieved 75%. On the other hand, recall which is true positives

over the predicted positives, label 0 yielded 74% and label 1 achieved 87%. Therefore, the f1 is 80% for both. The confusion matrix is also shown below.

```
from sklearn import metrics
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt


print(classification_report(label, pred, labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.87      0.74      0.80       752
           1       0.75      0.87      0.81       671

    accuracy                           0.80      1423
   macro avg       0.81      0.81      0.80      1423
weighted avg       0.81      0.80      0.80      1423
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt


cm = confusion_matrix(label, pred, labels=[0,1])

ax= plt.subplot()
sns.heatmap(cm, annot=True, fmt='g', ax=ax);  #annot=True to annotate cells, ftm='g' to disable scientific notation
# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['0', '1']); ax.yaxis.set_ticklabels(['0', '1']);
```



## 9. Model Inference

Once all the training and validation is done, the trained model is tested against the unseen data. First through a single sentence to see how it behaves and through the test dataset. The label 1 sentence has been predicted correctly.

```
#load weights
path='saved_weights2.pt'
model.load_state_dict(torch.load(path));
model.eval();

#Instantiate the spacy
import spacy
nlp = spacy.load('en_core_web_sm')
```

```
#define a function for prediction
def predict(model, sentence):
    '''Function that gives us prediction of the passed sentence
    Args: model, sentence-a string
    returns: Predictions'''
    sentence = review_to_words(sentence)
    #print(sentence)
    tokenized = [tok.text for tok in nlp.tokenizer(sentence)]  #tokenize the sentence
    #print(tokenized)
    indexed = [word_dict[t] for t in tokenized]          #convert to integer sequence
    #print(indexed)
    tensor = torch.LongTensor(indexed).to(device)
    #print(tensor)
    tensor = tensor.unsqueeze(1).T                              #reshape in form of batch,no. of words
    #print(tensor)
    prediction = model(tensor).argmax(dim=1)
    #print(prediction)
    return prediction.item()
```

```
#Lets use it for predicting the model

x = predict(model, "how the #altright uses  &amp;amp; insecurity to lure men into #whitesupremacy")
print(x)
```

```
1
```

The dataset was loaded, pre-processing done and sent through the prediction function. The resulting output is appended to the test dataset and written into a csv file.

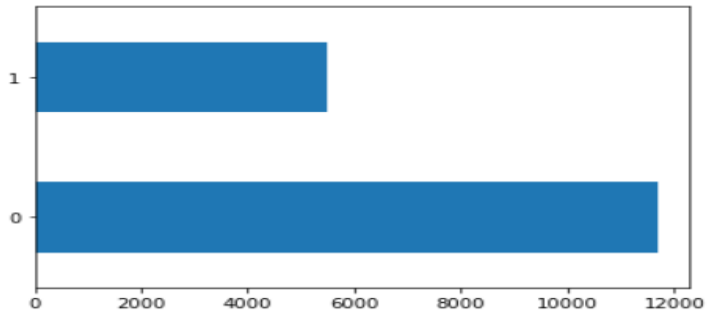
```
#we will now make a list of predictions
label_test = [predict_fn(model, df_test['tweet'][ind]) for ind in df_test.index]

print(label_test)
```

```
[0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0
0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0
1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0
0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1
1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0
1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1
0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0
0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0
1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0
```

The count plot shows us the number of 0 and 1 labels predicted in the test dataset along with the values.

```
#a simple visualization to show the prediction
df_test.label.value_counts().plot(kind='barh')
print(df_test.label.value_counts())
```

```
0    11710
1     5487
Name: label, dtype: int64
```




## 10. Conclusion

The twitter dataset had a raw text with so many noises. It has been processed, imbalance has been dealt and sent it through the feature extraction. Model was designed and training and validation was done. This model was able to predict 80% of the original tweets, that it had a hateful or non-hateful contents. It predicted 20% wrongly. It is a cyclic process where I must go back to text processing to do stemming and lemmatization, replacing the slang words with original English words, do a different balancing method and fine tuning the model would give me better results or might behave worse. But until I get much more efficient result, I must keep working on improving the model which is the goal.

## 11. Reflections on learning

Through this project, I had a wonderful opportunity to learn about Natural language processing, different types of text processing, feature extraction techniques. It added a great experience in my journey of Machine Learning and AI.

## 12. References


- A blog by Peter Bloem on Transformers, link.
- Text Classification using PyTorch by Analyticsvidhya, link.
- PyTorch Documentation, link.
- Attention is All you need, research paper by a google team, link
- Smote for Imbalanced classification method by Towards Data science, link.
- How to validate a model, Towards data science, link.