Classifying Types of Cyberbullying Found in Tweets

Taiwo Owoseni

Executive Summary

This project aims to build a predictive model by comparing accuracy of a machine learning model (random forest), deep learning algorithms (CNN, LSTM and Pretrained Transformer) to categorize tweets into cyber bullying classes and non-cyberbullying classes.

Link to project on github : here

Introduction

Cyberbullying is bullying with the use of digital technologies. It can take place on social media, messaging platforms, gaming platforms and mobile phones. It is repeated behavior, aimed at scaring, angering, or shaming those who are targeted[1].

In my contribution to social justice, I revealed past works done to flag tweets as cyberbullying and produce machine learning models that will address and identify tweets that are cyberbullying. It is my intention that this project could be adopted by Twitter to make users feel safe from bullies when they share their thoughts to the world.

Extensive research has been carried out and studied to identify cyberbullying. ThCNN-BiLSTM model with stacked word embedding was used by [4] to build a combined model that boosted the accuracy of cyberbullying detection. A pre-trained BERT model was used by [5] which is built on a novel deep learning network with the technique of transformer to detect cyberbullying on social media platforms.

Problem Definition

According to Comparitech's study on bullying data from 2018 – 2022, One-fifth of all bullying occurs through social media[2]. With the infamous attention cyberbullying gets through malicious misuse of freedom of speech on the internet to harm others, people's mental and emotional wellbeing are slowly deteriorating[3].

Reproducibility

For code reproducibility, seeds values are defined using python' s built in random function, torch and numpy. This is because these three libraries randomized data in different ways. Hence, it was computationally safer to use them together where randomization was performed in the code.

Dataset

The dataset used in this project was sourced from Kaggle[6]. This dataset consists of two columns: the tIet and cyberbullying type. The dataset contains 47,692 tweets. The table 1 below shows the count statistics of six classes in the dataset before and after data cleaning.

Cyber Bullying Type	Count	After Cleaning (Count)
Age	7,992	7,992
Ethnicity	7,961	7,959
religion	7,998	7,998

Not Cyberbullying	7,945	7,835
other cyberbullying	7,823	7,617
Gender	7,973	7,948
Total	**47,692**	**47,349**

Table 1: Cyber Bullying Types

Data Cleaning

A functional pipeline for effective data cleaning was built. The pipeline included the following:

a. Emoji Extraction

Emoji's were converted into their word equivalent. For instance, ❤️ was replaced with "red-heart" as text.

b. Profanity Extraction (Curse words)

I utilized data found on github [7] that listed curse words found in the tweets. I intended to use these words as feature data during model creation. I was unable to do so because some words like 'ass' are masked as 'a$$'. After punctuation removal, 'a$$' was transformed to 'a'. Due to this reason, I limited the use of profanity words to data analysis via plotting word clouds of profanity words in each class.

c. Hashtags and Mentions

I identified hashtags and mentions in the dataset. I removed mentions from the dataset and kept the hashtags because users tend to express themselves with hashtags. Hash symbol Ire replaced with whitespace.

d. Contraction

In the English language, writers tend to contract words like "is not" to "isn't". I found a function on github gist [8] that listed contractions and replaced them with their full words.

e. Links, Numbers, Punctuations and Stop words

Links, numbers, and punctuations are replaced with whitespace. NLTK English stopwords were used to remove all words that are considered stopwords. Additionally, words that are less than 3 characters are removed from each tweet.

f. Case Conversion and Text Trimming

All characters are converted to lower cases and stripped of extra whitespace from each tweet or record.

Data Splitting

The data was randomly splitted into three: Train, Validate and Test. Since the dataset is approximately balanced. The training portion contained 70 percent of the data and the validate and test portions contained 15 percent each. I validated the trained model on the dataset and evaluated how the models performed on "known" (I had the test data labeled) unseen data.

Models and Methods

a. Text Vectorization

I initialized a count vectorizer with its default training parameters from Scikit learn. I fitted and transformed the training data set on it, then generated feature data which became the new training data. To have a uniform number of

features in the validate and test data sets, I transformed them on the same count vectorizer.

b. Tokenization

I tokenized the training data using `Kera.Tokenizer()`. This class allows to vectorize a text corpus, by turning each text into either a sequence of integers (each integer being the index of a token in a dictionary) or into a vector where the coefficient for each token could be binary, based on word count, based on tf-idf[9].

The training data set was used as the corpus. I set a max_words to half the number of unique words in the corpus. This is because very few words occurred more than once. I fitted the Tokenizer on the corpus, converted the training, validation and test data to sequences. After which I padded any sequence that had a few number of vectors to ensure all the generated sequences had equal lengths.

c. Word Embedding

I generated an embedding matrix using the code I found on stack overflow[10] that used Glove to create word embedding. I use `nn.Embedding` to initialize a simple lookup table that stores embeddings of a fixed dictionary and size.This module is often used to store word embeddings and retrieve them using indices. The input to the module is a list of indices, and the output is the corresponding word embeddings [11]

I initialized the parameters to create the word embedding and did not train the weights when training the models.For modeling selection, the aim was to use simple explainable machine learning models then move to complex deep learning models. I implemented the following models Random Forest, CNN, BILSTM and Transfer Learning with a pretrained Hugging Face Transformer.

Models

In this section, the Random Forest model, I experimented and trained the data generated from the count vectorizer on the tree-based model. For two Deep Learning (CNN, BILSTM) models, I also experimented with one method for initializing word embeddings and text tokenization, which will be briefly above.

a. Random Forest Classifier

For the random forest classifier, the accuracy score was evaluated by cross-validation in five(5) folds. The best hyper parameter was selected based on the accuracy score when the model was fitted on the validation data. The entropy criterion and maximum depth of the best model were selected.

b. TextCNN

I used a convolution neural network because it extracts features by applying a filter to the dataset then finds patterns (new features to train on).

I Initializing the TextCNN, then create a list with different kernel sizes to be used during convolution. I initialize the embedding layer's Iight by generating parameters from embedding_matrix created earlier using gloves. I set the self.embedding.Iight.requires_grad` is set to False because I do not want to train the embeddings weights during model training.

I created a module list of four 2D convolution layers. For every layer, use an input channel of 1, an output channel that was equal to the number of filters and a kernel size that was generated from the list of filters created earlier. I set the dropout parameter to 0.1, for overfitting prevention, since I had a small set of training data. Finally, I created a forward pass that passed each convolution layer to a RELU activation layer, performed downsampling by dividing the input into 1-D pooling regions[12], then applied a linear layer to the final layer of a deep neural network is , the output, to applied to transform the output if the model into the number of classes I have.

c. Bi Directional LSTM

A Bidirectional LSTM (BiLSTM) is a recurrent neural network used primarily on natural language processing. Unlike standard LSTM, the input flows in both directions, and it's capable of utilizing information from both sides. It's also a powerful tool for modeling the sequential dependencies between words and phrases in both directions of the sequence[13]. This is the reason I decided to build a BILSTM.

I set the hidden size to 64 and the drop out to 0.1. I Initialize the embedding layer's Iight by generating a feature vector from embedding_matrix created earlier. I `self.embedding.weoght.requires_grad` is set to False because I don't want to train the embedding Lights. I Defined the LSTM model and set the bidirectional as True and batch_first as True. To use a bidirectional LSTM and to process that data in batches respectively.

In the forward pass, I set the hidden state as the embedding, I make the second hidden state the LSTM model. I generate the average and max pool for the second hidden state. I concatenate both layers and connect them to the next layer, the RELU, which is then connected to a dropout layer and finally, the output layer.

d. Hugging Face Pretrained Transformer

The pretrained (fine tuned) model used in this experiment was built using *DistilBertForSequenceClassification***.** The data was first tokenized using *DistilBertTokenizer* with a version that was built on lower cased words. During the model defintion and tuning, the class number was specified as 6, because this is the number of class to classify in this problem. Due to limited access to higher computational power , the default arguments of the *TrainingArgument* function were used. Finally, the default training arguments from *TrainingArgument* along with tokenized datasets()valid and train) and evaluation metric, were passed into the Trainer() function.

Results and Findings

To analyze the text data, I performed exploratory data analysis on the whole data. I created word clouds communicating the 50 most frequent words used in each class, and analyzed the distribution of average words per Cyberbullying Type. In image 9, the ethnicity cyberbullying type has the highest number of curse words.

Additionally, the cyberbullying type religion has the highest number of word counts.

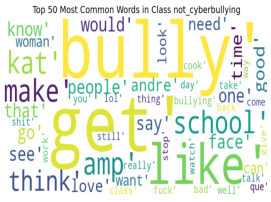

Image 1: Top 50 Most Common Words in Class Not Bullying

Image 2: Top 50 Most Common Words in Class Not Bullying

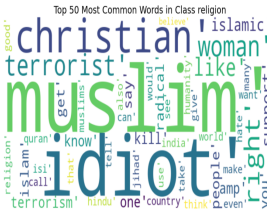

Image 3: Top 50 Most Common Words in Class Religion

Image 4: Top 50 Most Common Words in Class Other CyberBullying

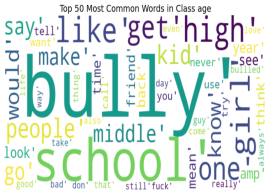

Image 5: Top 50 Most Common Words in Class Age

Image 6: Top 50 Most Common Words in Class Ethnicity

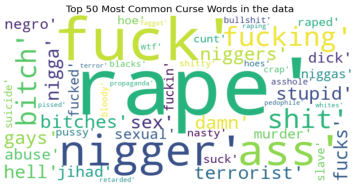

Image 7: Top 50 Most Common Curse Words in the data

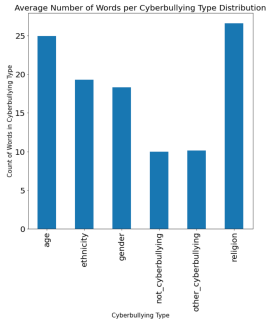

Image 8: Average Number of Words per Cyberbullying Type Distribution

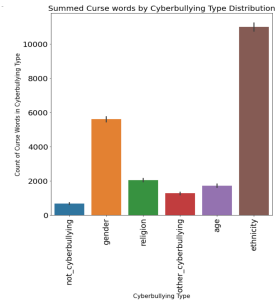

Image 9: Summed Curse words by Cyberbullying Type Distribution

a. Confusion Matrix and Valid/Train Loss Curve

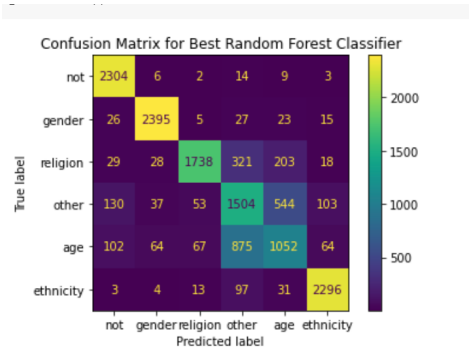

Image 10: Random Forest confusion Matrix

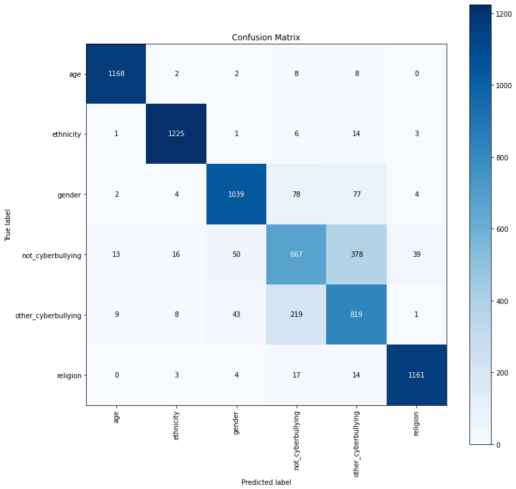

Image 11: CNN Confusion Matrix

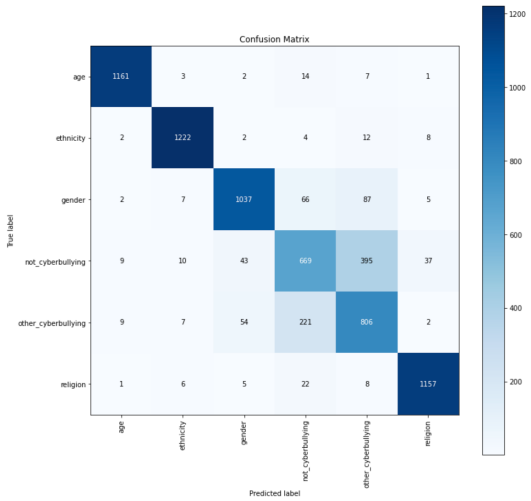

Image 12: BI-LSTM Confusion Matrix

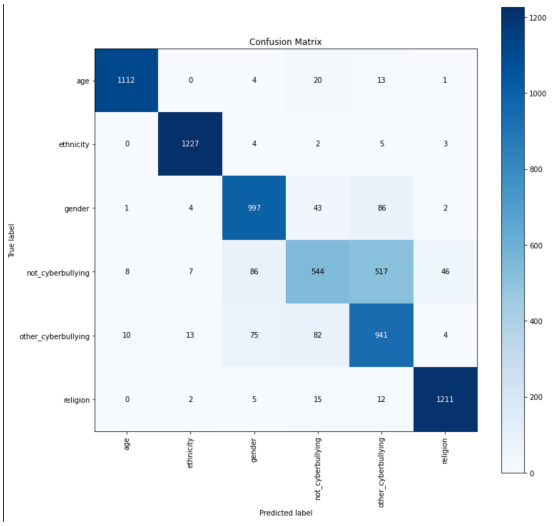

Image 15: Hugging Face fine tuned Transformer Confusion Matrix

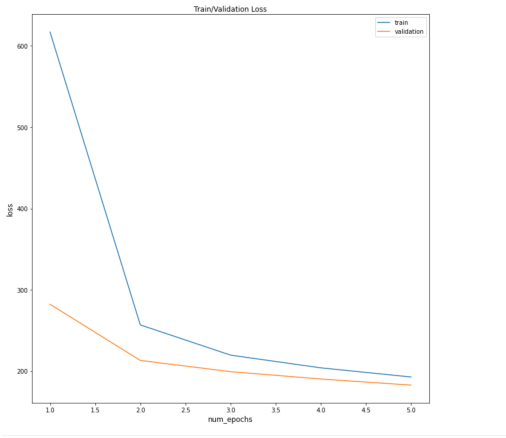

Image 13: BI-LSTM training loss curve

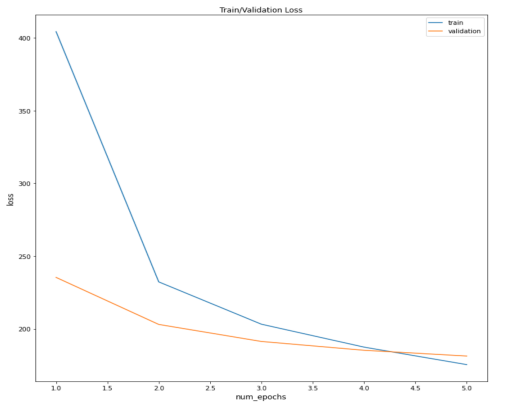

Image 14: CNN training loss curve

Models	Validation Accuracy	Test Accuracy
Random Forest + CountVectorizer	0.7868	0.7947
BI-LSTM + Glove Embedding	0.8518	0.8399
CNN + Glove Embedding	0.8558	0.84060
DistilBertForSequence Classification + DistilBertTokenizer (Fine tuned)	None	0.8493

Table 2: Model Training Accuracy and Validation score

Conclusions and future work

This experiment was performed on a Machine learning model, two deep learning models and fine tuned transformers. The Fine tuned model performed best with a test accuracy of 84.93%. Although there were high expectations for the fine-tuned transformer, its performance could be made better by

setting more hyperparameters for training. Additionally, other word embedding methods like word2vec could be used for word embedding and vectorization.

Future works aim to create a grid of various tokenizers, word vectorized and transformers they compare the validation and training accuracy. Due to the high computational power of the deep learning and transformer models, k-fold splitting was not carried out. Future work should include this step. Additionally, the training and validation loss curve was not omitted for the fine tuned transformer model, this should be included in future works.

Reference

1. https://www.unicef.org/end-violence/how-to-stop-cyberbullying
2. https://www.comparitech.com/internet-providers/cyberbullying-statistics/
3. https://www.healthshots.com/mind/mental-health/3-ways-in-which-cyberbullying-can-impact-your-mental-health/
4. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9321314/
5. https://www.semanticscholar.org/paper/Cyberbullying-Detection-using-Pre-Trained-BERT-Yadav-Kumar/3755d340754579beeac8ee1cd8217a72b1c8682c
6. https://www.kaggle.com/datasets/andrewmvd/cyberbullying-classification
7. https://github.com/zacanger/profane-words/blob/master/words.json
8. https://gist.github.com/MLWhiz/a603656c482ce13f3f2affc1d35f287d
9. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer
10. https://www.kaggle.com/code/gmhost/gru-capsule/notebook
11. https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html
12. https://www.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.maxpooling1dlayer.html
13. https://www.baeldung.com/cs/bidirectional-vs-unidirectional-lstm