TOXIC COMMENT DETECTION USING DEEP LEARNING ALGORITHM

Aditi Shrivastava

Abstract

The internet provides a public forum where anyone can openly discuss any topic. On the other hand, harassment and abuse are deterring people from voicing their opinions and upsetting the online community. Because sites find it difficult to successfully guide conversations, many communities choose to restrict or remove offensive user comments.Therefore, it can be challenging to talk about the topics you care about. Many people give up on finding out alternative viewpoints and cease expressing themselves online due to the fear of abuse and harassment. Many communities restrict or disable user comments because platforms are unable to successfully foster conversations.

This issue drives our desire to develop deep learning models that can recognize toxicity during online exchanges, which is defined as anything that is unpleasant, disrespectful, or likely to drive someone away from a debate. Our goal is to use deep learning models to create technology that can protect voices in conversations. This project used concepts of Bidirectional Long Short-Term Memory (Bi-LSTM) along with concepts of Embedding to design the sequential model. For the preprocessing of the text data, TextVectorization is used. TextVectorization facilitates working with text data in TensorFlow easier by streamlining the preprocessing of text information to feed neural network models. Alongside, the libraries used for this project are pandas, tensorflow, numpy, keras and for plotting, matplotlib is used.

This model reads the training csv which divides a statement into 6 different categories: toxic, severe_toxic, obscene, threat, insult, identity_hate and is trained using the deep learning concepts. The accuracy obtained for the training set and validation set in this model is 99%.

DataSet:

For training and testing of the model, csv files are taken from Kaggle. The training set has almost 160K data which are categorised into a table with columns as toxic, severe_toxic, obscene, threat, insult, identity_hate.If all the columns are zero, then the statement is considered as non-toxic else, the column under which the statement fits well is considered as one.

Description of chosen DL algorithm(s)

This model is a sequential neural network design, most commonly employed for a sequence-based classification or prediction problem. Let's examine each layer in turn:

Bi-LSTM: Long Short-Term Memory, or LSTM, is a kind of RNN that was created to get around traditional RNNs' shortcomings in identifying long-term relationships in sequential data. Memory cells and gating mechanisms are included to allow for the gradual selective retention and forgetting of knowledge. Because of their long-lasting internal memory state,

long-term dependencies spanning multiple time steps can be captured by LSTMs. Bi-LSTM concurrently processes the sequence in both ways, in contrast to conventional Recurrent neural networks (RNNs) which merely evaluate sequences of inputs in a single direction (either ahead or backward). It is made up of two LSTM layers, one of which processes the sequence forward and the other backward. Every layer preserves its own memory cells and hidden states.
The series of inputs is introduced to the forward-looking LSTM layer starting at the initial phase and continuing through the end of the forward pass. Based on the present input, the prior hidden state, and the memory cell, the forward-looking LSTM determines its hidden state and modifies its memory cell at each successive step. In addition, the input sequence is simultaneously supplied through the backward LSTM tier from the final time step to the first, in reverse order. The reverse LSTM determines the hidden state and modifies its memory cell depending on the input received today as well as the prior hidden state as well as memory cell, just like the forward pass does.The hidden states across both LSTM layers are integrated at each time step when the forward and backward passes are finished. Combining the hidden states and using another transformation can be all that is needed to create this combination.

Embedding: With word embeddings, we can employ a dense, effective representation where words with comparable encodings are used interchangeably. Crucially, you do not need to manually define this encoding. A dense vector of floating point values is called an embedding; the vector's length is a parameter that you set. The values for the embedding don't need to be specified explicitly since they are trainable parameters—weights that the model learns during training, much like it learns weights for a thick layer. Word embeddings with 8 dimensions (for small datasets) or up to 1024 dimensions (for large datasets) are frequently seen. Fine-grained word associations can be captured by a higher dimensional embedding, but it requires more data to train.

Simplified review of the approach and process used to perform the project.

This project can be broadly divided into 5 steps, namely preprocess, creating sequential models, making predictions, evaluation of the model and finally testing the model.

Preprocessing: In this stage, the dataset taken from Kaggle dataset divides comments into six categories: toxic, severe_toxic, obscene, threat, insult, identity_hate. When there are only zeros in the seventh category, it indicates that the comment is not harmful. Next, we can create a single category out of the six. The information in the database now only indicates whether a comment is harmful or not. Thus, our issue is merely one of binary classification.

For Preprocessing of the dataset, the concept of TextVectorization from Tensorflow is used. The vocabulary's maximum size is determined by the max tokens parameter. It indicates how many distinct tokens (words or subwords) the TextVectorization phase is going to monitor to a maximum extent. For this project the max_token is set to 200,000.

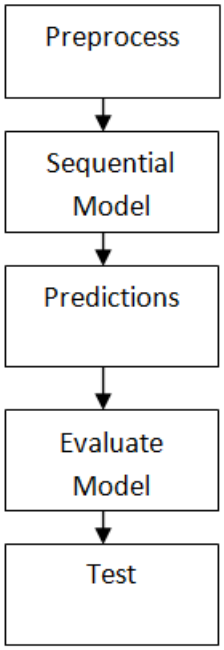


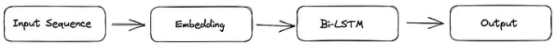
Fig1. High level diagram about your solution (showing individual components).

Sequential Model:

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

 bidirectional (Bidirection  (None, 64)                16640
 al)

 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: 6491686 (24.76 MB)
Trainable params: 6491686 (24.76 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Fig2. The Sequential Model Layer-wise Breakdown

The above framework uses an arrangement of dense layers to anticipate words based on variable-length word index sequences that have been processed through bidirectional and embedding layers to obtain contextual data. The framework's output, which is likely a representation of the likelihood for each category in a task involving classification, is given by the last layer.



Input Sequence: A series of information points, such as phrases in an expression or symbols in a text, make up the input sequence. Vector or embedding representations are commonly used to represent individual data points.

Embedding: Embeddings are dense vector representations created by often transforming the input sequence. In addition to giving the ensuing layers a more condensed and insightful representation, embeddings additionally record the semantic significance of each of the information pieces.

Bi-LSTM:The essential element of the architecture is the Bi-LSTM layer. It is made up of a pair of LSTM layers, the first of which processes the input sequence forward and the other backward. Every layer of an LSTM has a unique set of parameters.

Prediction and Evaluation:

For the evaluation, the batch is unpacked and then the prediction is made, later flatten it. Keras metrics such as prediction, recall and CategoricalAccuracy is used to evaluate the model. Input text is provided to predict the model.

Testing:

The function known as score_comment receives an expression as input from the user, prepares it utilizing a TextVectorization layer (vectorizer),

supplies the machine learning model (model) with the preprocessed comment, and structures the model predictions to produce a text-based output. The score_comment method is utilized in order to offer predictions for the comments entered by users via an interface that has been designed for that purpose. The forecasts will appear as text on the interface. After that, interface.launch() is used to open your browser's interface locally.

Result and Conclusion

One advantage of Bi-LSTM is that, unlike typical RNNs, it can record both the context preceding and following a given time step. Richer interdependence in the input sequence can be captured by Bi-LSTM by taking into account both historical and prospective information.

The accuracy obtained using the Embedding and Bi-LSTM sequential model for validation and training set is around 99% and the loss is around 0.13.

Fig3. Plot of Loss vs Epoch.

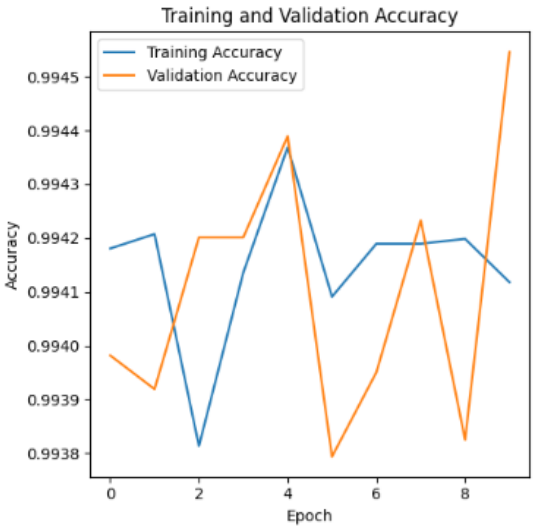


Fig4. Plot of Accuracy vs Epoch

The Gradio library is used to generate the interface for the testing of the model. The interface has two sections: first one is the comment and the second is the output.

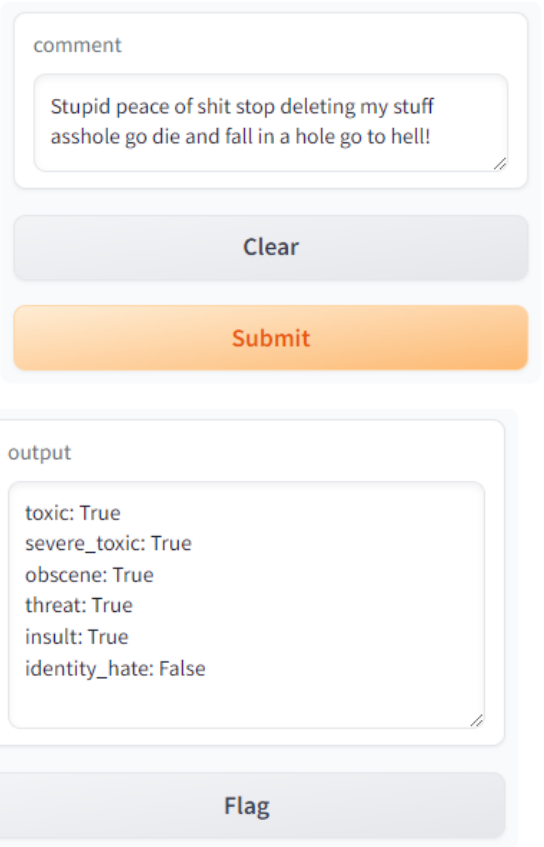


Fig5. The interface where the comment written is toxic and the output reads it as toxic.

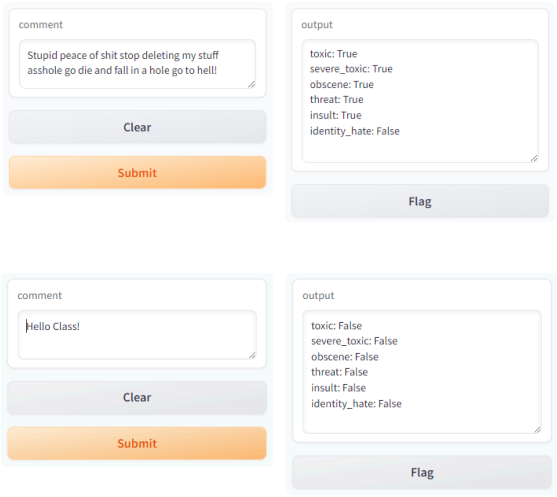


Fig6. a) The interface where the comment written is toxic and the output reads it as toxic.b) The interface where the comment written is ***Hey Class!*** and the output reads it as non-toxic (if all the category is displayed as False, which mean the comment is non-toxic)

Future Endorsement

With additional time and resources, would have performed more testing such as increasing the epoch size and testing on a csv file. The GPU usage was limited , so if considered the knowledge and resource could have been tested on a later set.

Also,would have worked on cutting-edge architectures created especially for NLP (natural language processing) applications, such as the categorization of comment toxicity and also, would have taken into account designs like Transformer-based models (BERT, RoBERTa, GPT, etc.) or variations on them that have been optimised for toxicity detection applications. Also, for the further study on comment toxicity, will examine ensemble approaches, which integrate models' predictions to enhance performance and for scaling the instruction across numerous GPUs or workstations, use distributed training methodologies.

For large-scale distributed learning across several instances, utilise cloud-based distribution learning platforms like Google Cloud AI Platform or AWS Distributed Training could be used to make the training dataset more diverse, use sophisticated text-specific data augmentation techniques. Methods like contextual augmentation, paraphrase, and back-translation can assist strengthen the stability of the toxicity classification model.

References

1. Kevin Khieu, and Neha Narwal Detecting and Classifying Toxic Comments
2. Sara Zaheri, Jeff Leath and David Stroud Toxic Comment Classification
3. Alissa de Bruijn, Vesa Muhonen, Tommaso Albinonistraat, Wan Fokkink, and Peter Bloem. Detecting offensive language using transfer learning
4. Spiros V. Georgakopoulos, Sotiris K. Tasoulis, Aristidis G. Vrahatis,Vassilis P. Plagianakos Convolutional Neural Networks for Toxic Comment Classification
5. Betty van Aken, Julian Risch, Ralf Krestel, Alexander Löser Challenges for Toxic Comment Classification: An In-Depth Error Analysis
6. Viera Maslej-Krešňáková ,Martin Sarnovský, Peter Butka and Kristína Machová Comparison of Deep Learning Models and Various Text Pre-Processing Techniques for the Toxic Comments Classification
7. Pallam Ravi, Hari Narayana Batta, Greeshma S, Shaik Yaseen Toxic Comment Classification