JIGSAW Multilingual Toxic Comment Classification

Made By : Bishoy Adel And Youssef Magdy

September 11, 2024

Introduction

This project aims to build a model for toxic comment classification using the JIGSAW dataset. The challenge involves understanding toxic language in multiple languages and handling the nuances of language differences. We will approach the task in five distinct phases.

Phase 1: Data Exploration and Preprocessing

In this phase, we will focus on understanding the dataset and preparing it for modeling.

1.1 Data Analysis

The JIGSAW multilingual dataset consists of different splits across languages. The training set is composed exclusively of English comments, while the validation set includes comments from three languages: Spanish, Italian, and Turkish. The test set, on the other hand, contains comments from six languages: Spanish, Italian, French, Portuguese, Russian, and Turkish. This setup introduces a multilingual challenge for the model, requiring it to generalize across multiple languages despite being trained on English-only data.

```
print(f"Train Shape is  {train.shape[0]}")
print(f"Validation Shape is  {validation.shape[0]}")
print(f"Test Shape is  {test.shape[0]}")
```

```
Train Shape is  223549
Validation Shape is  8000
Test Shape is  63812
```

Figure 1: The Number of examples in the given dataset.

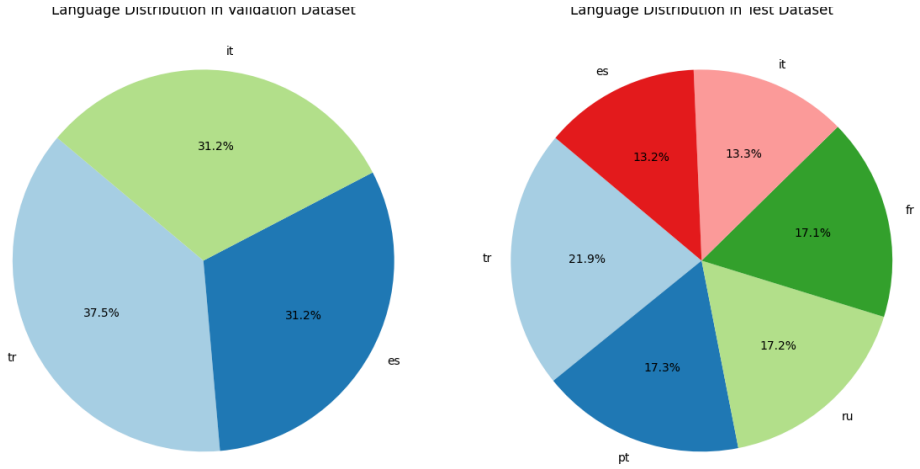


Figure 2: Distribution of Validation and Test sets.

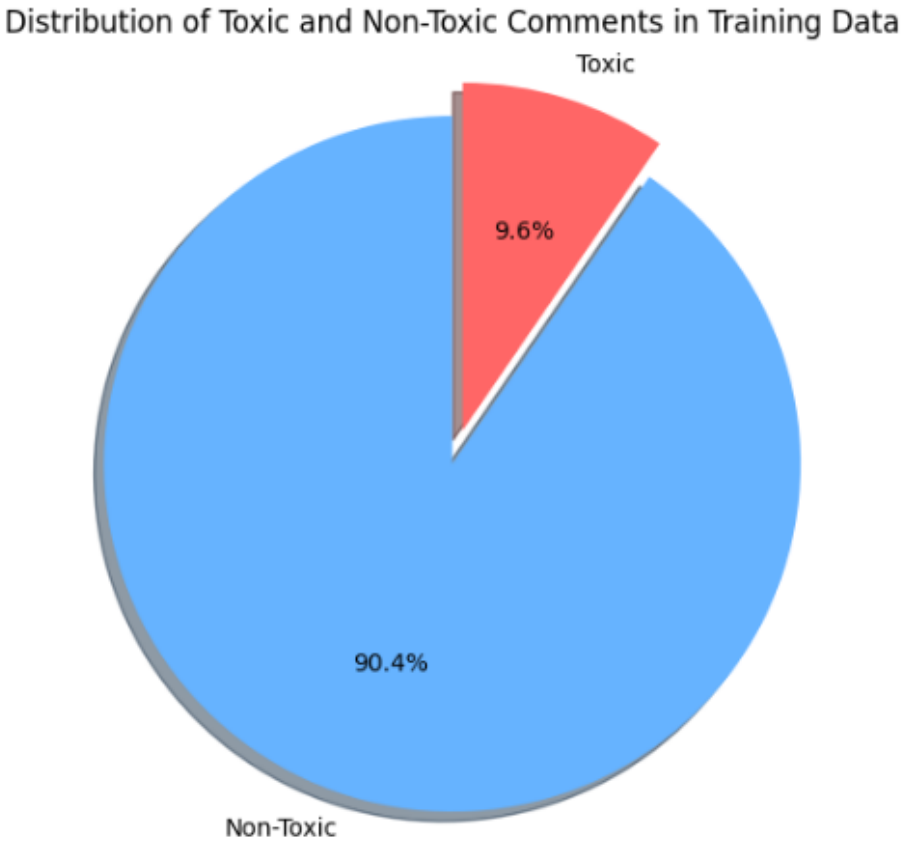


Figure 3: Distribution of Toxic and Non-Toxic Comments in Training Data.

1.2 Data Preprocessing

For our translation process, we utilized a dataset created with the Google Translate API. This dataset includes translations of English comments into several target languages, such as French, Spanish, Turkish, Russian, Italian, and Portuguese. By employing this pre-translated dataset, we effectively prepare a multilingual corpus that reflects the linguistic diversity of our validation and test sets. To manage resources efficiently and handle interruptions, we process the dataset in batches of 100 comments and save the results incrementally to a CSV file.

1.3 Evaluation Metric

The evaluation metric we used was Area under ROC curve because the ROC curve and AUC are crucial for evaluating model performance in imbalanced datasets because they provide insights into how well the model can distinguish between classes across all decision thresholds, rather than just measuring overall accuracy. This makes them more reliable metrics when dealing with imbalanced data where the class distribution is skewed .

Phase 2: BaseLine Model Development

Here, we will focus on designing and implementing the baseline model.

2.1 BaseLine Model Tokenizer

The word-level tokenizer from Keras was used to preprocess the text data. It tokenizes the text by converting words into integer sequences based on their frequency in the dataset. We set the maximum sequence length (`max_len`) to 2400, as the longest comment in the dataset had 2321 characters. This preprocessing step ensures that each comment is transformed into a numerical format suitable for model input. Additionally, padding was applied to the sequences, so all inputs have the same length, allowing the model to process them uniformly.

2.2 Model Architecture

The baseline model is built using a Simple Recurrent Neural Network (RNN). The architecture includes an embedding layer to convert the input tokens into dense vectors with embedding size 300 , followed by a Simple RNN layer to capture sequential patterns in the data. Finally, a dense layer with a sigmoid activation function is used to perform binary classification, predicting whether a comment is toxic or non-toxic. The model is optimized using binary cross-entropy and the Adam optimizer, ensuring efficient training.

2.3 Model Evaluation

In our case, the primary evaluation metric used was the ROC-AUC (Area Under the Receiver Operating Characteristic Curve). After translating the data into the six languages present in the test set (Spanish, Italian, French, Portuguese, Russian, and Turkish), we faced resource limitations. To address this, we dropped 50% of the translated comments from each language while keeping the English comments intact.

We then retrained the same Simple RNN model with the same architecture. The Area Under the ROC Curve (AUC) for this experiment was 0.7896, reflecting the model's ability to distinguish between toxic and non-toxic comments across multiple languages.

Phase 3: Second Model Development

Here, we will focus on designing and implementing more robust solution.

3.1 Model Tokenizer

For the second model, we used the `DistilBertTokenizerFast` from the Hugging Face transformers library. This tokenizer is pre-trained for multiple languages and efficiently handles tokenizing multilingual text. It splits text into subword tokens, ensuring that even uncommon words are represented adequately. The tokenizer also pads and truncates sequences to match the input length required by the model, ensuring consistent input dimensions across batches.

3.2 Model Architecture

The model architecture was based on the `distilbert-base-multilingual-cased` pre-trained model which is trained on 104 languages. DistilBERT is a lighter and faster variant of BERT, designed to be computationally efficient while retaining much of BERT's accuracy. For this setup, we used a binary classification layer on top of DistilBERT to predict whether comments were toxic or not. The loss function used was binary cross-entropy, and the optimizer selected was Adam, with a starting learning rate of `1e-5`. This configuration allowed for stable convergence during training while keeping the model efficient but we faced some problems which was that the dataset size was very big so we trained the model for 2 epochs for about 11 hours but we found a great method that utilized the performance in the final model which was the distributed datasets.

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_word_ids (InputLayer)  [(None, 192)]             0
_________________________________________________________________
tf_distil_bert_model (TFDist ((None, 192, 768),)       134734080
_________________________________________________________________
tf_op_layer_strided_slice (T [(None, 768)]             0
_________________________________________________________________
dense (Dense)                (None, 1)                 769
=================================================================
```

Figure 4: Model Architecture.

3.3 Model Evaluation

The model was trained for two epochs, and the primary evaluation metric used was the Area Under the ROC Curve (AUC). After training, we achieved an AUC score of 0.8694, indicating a significant improvement in the model's ability to distinguish between toxic and non-toxic comments across multiple languages.

Figure 5: Evaluation Metric.

Phase 4: Final Model Development

Here, we will focus on designing and implementing the Final model.

4.1 Preprocessing Steps Taken

As a Preprocessing step we first used the translated dataset to the 6 languages using the Google API and then we downsampled the number of comments where the number of toxic comments should be equal to the number of non-toxic comments so we had 256496 multilingual (balanced) comments to train on .

After That we used distributed datasets which are commonly used for efficient data input pipelines in TensorFlow, especially when training models on TPUs. When training with multiple devices such as TPUs, data needs to be distributed across replicas for parallel processing, and that's where tf.distribute comes in. A DistributedDataset is an automatically managed dataset that distributes the data across multiple replicas.

Fine tuning NLP transformer models on the MLM task was found to improve the performance on many downstream tasks such as classification , so we used an xlm Roberta large model fine tuned for MLM on the test data. Same approach used by first and third place ranking teams in the competition

]:

	comment_text	toxic
40468	24/7\n\nArrêtez d'apporter des modifications i...	1
46720	"'mejorar la suerte de los pobres, pero afianz...	0
46202	Heil Hitler! Heil Hitler! Heil Hitler! Heil Hi...	1
56165	":: Cher Samboy, je ne crains pas du tout les...	0
144634	J'ai ajouté l'invasion de l'Irak en 2003, car ...	0
57729	Me gustaría trasladar a los protestantes del U...	0
88100	Şimdi kadeh kaldırıyoruz\n\nha ha öylesine tos...	1
13301	Красная ссылка\n\nВы недавно заново создали сс...	0
67258	¡Maldito infierno!\nEres tan idiota. ¡Deja de ...	1
29805	Muy buen contribuyente? Mierda. La camarilla n...	1

Figure 6: Example from balanced training dataset after downsampling

4.2 Model Tokenizer

For this project, the tokenizer used is `jplu/tf-xlm-roberta-large`, which is a multilingual pre-trained model. The tokenization is handled by the `regular_encode` function, which takes a list of texts and encodes them using the pre-trained tokenizer. The function uses the `batch_encode_plus` method, ensuring that the tokenized sequences are padded to a maximum length of 512 tokens, with

padding added as necessary. The encoded input IDs are returned as NumPy arrays, which are then used as inputs for training and validation datasets. Specifically, the comments from the training, validation, and test sets are tokenized and stored , facilitating the training process for the multilingual toxic comment classification model.

4.3 Model Architecture

The model used for multilingual toxic comment classification is based on the `tf_roberta_model`, a pre-trained transformer-based model from the `TFBaseModelOutputWithPoolingAndCrossAttentions` class. The following table summarizes the architecture of the model:

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_word_ids (InputLayer)  [(None, 192)]            0

 tf_roberta_model (TFRoberta  TFBaseModelOutputWithPoo  559890432
 Model)                      lingAndCrossAttentions(l
                             ast_hidden_state=(None,
                             192, 1024),
                              pooler_output=(None, 10
                             24),
                              past_key_values=None, h
                             idden_states=None, atten
                             tions=None, cross_attent
                             ions=None)

 tf.__operators__.getitem (S  (None, 1024)            0
 licingOpLambda)

 dropout_73 (Dropout)        (None, 1024)              0

 custom_head (Dense)         (None, 1)                 1025

=================================================================
Total params: 559,891,457
Trainable params: 559,891,457
```

The model consists of an input layer receiving word IDs with a sequence length of 192, followed by the RoBERTa model, which outputs a last hidden state with dimensions of (192, 1024) and a pooled output of size 1024. The final custom dense head outputs a single value indicating the toxicity of the comment.

We added a droput layer with 0.5 probability and a dense layer with one unit

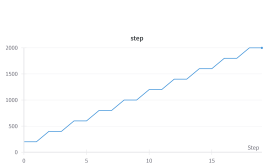
and a sigmoid activation on top of the pretrained model mentioned , also Used two different learning rates for the classification head and the pretrained model respectively as we used smaller learning rate for the pretrained model and the higher one for the new untrained dense layer.

4.4 Model Evaluation

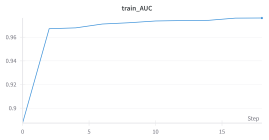
The model was trained for two stages , the first stage was with the training data listed above and the second one was with the validation data , and the primary evaluation metric used was the Area Under the ROC Curve (AUC). After training, we achieved an AUC score of 0.94143, indicating a significant improvement in the model's ability to distinguish between toxic and non-toxic comments across multiple languages , Known that the first place in this competition got Area under the ROC curve = 0.9537 .
And here are some plots to show the metrics we were watching during both stages using WANDB to generate all these important plots .

Stage 1

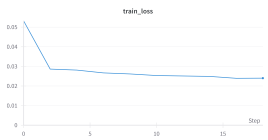
for the first training stage we used the downsampled translated training data discussed earlier ,Ran it for 2000 training steps , and then used wandb library to track metrics such as training and validation loss and AUC score.



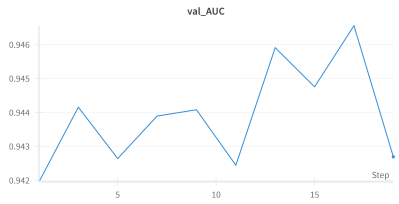
(a) No of Steps



(b) Training area under ROC curve



(c) Training Loss



(d) Validation area under ROC curve

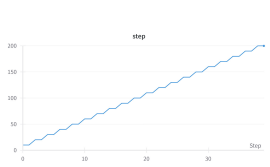
(e) Validation Loss

Figure 7: Metrics of Stage 1.

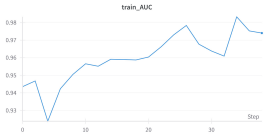
Stage 2

Translated data may be a little bit misleading because sometimes the sentiments don't transfer across different languages ,we used the validation set for stage II training , which consisted of real Foreign language comments from three different languages to fine tune the model for 200 training steps.

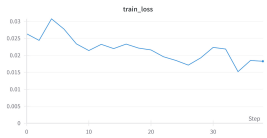
This resulted in about an 0.004 boost in the final score , since the validation set is relatively small (only 8000 samples)



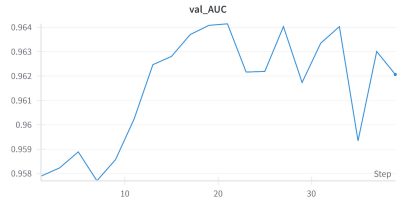
(a) Number of Steps



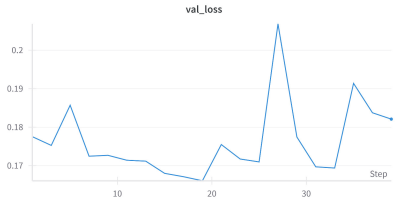
(b) Training Area Under ROC Curve



(c) Training Loss



(d) Validation Area Under ROC Curve



(e) Validation Loss

Figure 8: Metrics of Stage 2.

The final output Area Under the ROC Curve on the test data, which contains all the 6 multilingual data:

In [17]:
```
from sklearn.metrics import roc_auc_score
ytrue = pd.read_csv(D+'test_labels.csv')['toxic']
print(roc_auc_score(ytrue, sub_df['toxic']))
```

```
0.9414304246027457
```

Figure 9: ROC Score for Multilingual Data

Phase 5: Final Model

Here, we will focus on changing and modifying the Final model.

5.1 Modifications made

As we discussed in the previous section that we used an xlm Roberta large model fine tuned for MLM on the test data.In this section we will try to fine tune the model on the training balanced multilingual data we made using the same downsampling technique discussed in section 4.1 to remove any bias for the model in seeing the test data before predicting then we will compare the results

Here we harness the power of distributed datasets to handle the computationally intensive task of Masked Language Modeling (MLM). Given the scale and complexity of MLM, which requires predicting missing words in sentences and learning deep contextual relationships, the training process can be time-consuming. By leveraging distributed datasets, we can efficiently partition the workload across multiple nodes, allowing for parallel data processing and faster training times. This distributed approach not only enhances the model's scalability but also ensures that the massive data needed for MLM tasks is managed effectively, reducing bottlenecks and improving overall performance.

5.2 Model Architecture

The architecture used for masked language modeling is based on the **TFXLM-Roberta** model, a variant of the XLM-Roberta model designed specifically for masked language modeling tasks. The model consists of two main components: the **roberta** layer and the **lm_head** layer. The **roberta** layer, which contains the core transformer blocks, has a total of 558,840,832 parameters. This layer is responsible for encoding the input sequence into high-dimensional representations. On top of this, the **lm_head** layer, responsible for predicting masked tokens, consists of an additional 257,833,106 parameters. In total, the model comprises **560,142,482 parameters**, all of which are trainable. The model occupies approximately 2.09 GB of memory, with all parameters being actively used during training, as indicated by the absence of non-trainable parameters.

```
Model: "tfxlm_roberta_for_masked_lm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 roberta (TFXLMRobertaMainL  multiple                  558840832
 ayer)

 lm_head (TFXLMRobertaLMHea  multiple                  257833106
 d)

=================================================================
Total params: 560142482 (2.09 GB)
Trainable params: 560142482 (2.09 GB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 10: Model Architecture

5.3 Evaluation Metric

After fine tuning the roberta on the training data not the test data to remove any bias the Area under ROC curve was 0.9397 .

```
In [17]: from sklearn.metrics import roc_auc_score
         ytrue = pd.read_csv(D+'test_labels.csv')['toxic']
         print(roc_auc_score(ytrue, sub_df['toxic']))

         0.9397015171682328
```

Figure 11: Area Under ROC Curve

Architecture	**Area under ROC curve**
Simple RNN	0.7896
BERT 1	0.8694
XLM ROBERTA LARGE (MLM on test)	0.9414
XLM ROBERTA LARGE (MLM on training)	0.9397

Table 1: Comparison of the results