Analyzing the Contextual Performance of Machine Learning vs Transformers in Toxic Comment Classification

1) Aditya Shanmugham (NUID : 002738073)

2) Gugan Kathiresan (NUID : 002756523)

3) Maria Anson (NUID : 002931419)

Abstract

Toxic comment classification is a significant niche in the field of natural language processing owing to its vast real-world applications. Popularly used in the field of digital media, toxic comment classification is vital towards maintaining a decorous and peaceful society. The use of natural language processing algorithms to accelerate toxic comment classification across various data sources, contextual situations and languages have been attempted by researchers of late. This study proposes a comparative analysis of fundamental machine learning algorithms and paradigm transformer architectures to classify multilingual toxic comments. The methods employed in this study include k-nearest neighbors, xgboost and transformers architectures like BERT, etc. These methods are compared to verify their performance and efficiency in a contextual manner. From the results obtained, it was inferred that transformers offer better quality classification compared to the basic machine learning models. The ability of transformers to handle multilingual data makes it a viable solution that can be deployed in real-world applications. Future improvements could be to deploy the proposed model for real-time computations.

1. Introduction

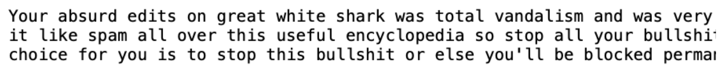
The rise of social media platforms has enabled people to express their thoughts and opinions publicly. Communicating your messages and views in a public forum has become as easy as a click. This has led to an increase in the number of comments online. In the year 2023, over 500 million tweets are sent out per day on Twitter [1]. These comments are not always constructive and can sometimes contain toxic language. Identifying these comments and their levels of toxicity is a herculean task. Toxic comments can be harmful to individuals and communities, and they can also negatively affect online conversations.

This project proposal aims to develop an NLP model that can detect toxic comments. The ultimate goal of this project is to identify the optimum method that can provide contextually relevant classifications of higher quality by comparing performances of fundamental algorithms (machine learning models) to trending architectures right now (transformers) and analyze the performance.

Toxic comments are complex to identify. They are characteristically described by their toxic vocabulary and their targeted context. Figure 1.1 and 1.2 are examples of a toxic and non-toxic comment from the dataset details in section 5.1. However, the complexity arises when these contexts are combined with the slang or a sentence structure native to a linguistic demographic. These instances require the use of natural language processing algorithms to encode and decode these toxic comments.

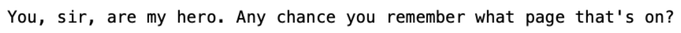
A popular approach employed by recent works is the use of transformer models. Transformer models are popular due to their ability to effectively model long-range dependencies in sequences of data, such as text, audio, and images. This is achieved through the use of self-attention mechanisms, which allow the model to focus on different parts of the input sequence during each step of the computation. Additionally, transformer models have shown impressive results in a wide range of natural language processing tasks, including machine translation, language modeling, and question answering. As a result, transformer models have become a cornerstone of modern deep learning research and have significantly advanced the state of the art in various domains.

Recent works that aimed to classify toxic comments on a multilingual basis include employing LSTMs [2,3], and even pre-trained Convolutional Neural Networks [4, 5], with some works employing transformer architectures as well [6]. We take inspiration from these works to create classification models but yet improve them with transformers.



```
Your absurd edits on great white shark was total vandalism and was very
it like spam all over this useful encyclopedia so stop all your bullshi
choice for you is to stop this bullshit or else you'll be blocked perma
```

Figure.1.1 Toxic Comment



```
You, sir, are my hero. Any chance you remember what page that's on?
```

Figure.1.2 Non Toxic Comment

The following sections can be briefly be categorized as a description of the problem statement (Section 2), the methodology employed (Section 3), a short explanation on the theory behind the proposal (Section 4), the details involving the experiments conducted in the study (Section 5) and the summary of our findings with discussion (Section 6).

2. Problem Statement

Research studies in toxic comment classification have statistically been focused on the development of novel models with the help of transfer learning [2-6]. However, there is room for expansion of these models to a multilingual dataset. In this research, we conduct comprehensive study over several well known supervised learning algorithms. Through this, we attempt to understand the complexity behind multilingual datasets, the best suited model and its application to toxic comment classification.

3. Methodology

A wide and detailed multilingual toxic comment database was obtained from Kaggle and analyzed. The data was preprocessed to binary label and then normalized by each column and mapped to [0,1]. Then 70% of the data points are selected for training and 30% are selected for testing.

4. Theory

The proposed model, be it the machine learning model or the transformer architecture, is predominantly used to handle the feature extraction. Currently, the dataset at hand is well annotated and available preprocessed for transformer models. The datasets have been constructed as a binary classification problem with train, validation and testing splits. Nevertheless we will analyze the dataset for any imbalances or bias.
The three approaches being compared in this project include:

I. Approach 1 : Machine Learning Models (KNN, Random Forest, Xgboost)
II. Approach 2: Transformers with feed forward
III. Approach 3: Stacked transformers.

4.1 Approach 1: Machine Learning Models

4.1.1. Data Split

The process involved combining the train and validation datasets to form a unified dataset, which was subsequently divided into training and validation subsets using an 80/20 split. The test data was merged with an inner join to obtain the corresponding test labels, utilizing a common identifier as the linking key.

4.1.2. Data Preprocessing

For data preprocessing, the training, validation, and test sets underwent cleaning procedures such as removal of numbers, common stop words, and extra white spaces. Additionally, lemmatization was performed. Subsequently, the text data was encoded using TF-IDF embedding, which assigns more weight to words that appear frequently in a document but less frequently in the entire corpus. This technique is particularly useful for representing toxic words since they are more informative in identifying the content of a particular document.

4.1.3. Model Selection

1. KNN (K Nearest Neighbour) : KNN was chosen because it takes into consideration the closest neighbors, which can potentially reveal patterns in the data, particularly if toxic sentences appear closer to each other in the vector representation.
2. Random Forest Classifier : Random Forest is chosen for its ability to handle large datasets, low risk of overfitting, feature importance analysis, robustness to outliers, and versatility make it a popular choice for classification tasks with complex and noisy data.
3. Xgboost Classifier : Xgboost often achieves higher accuracy, is highly scalable, has advanced optimization capabilities, can prevent overfitting with regularization and takes advantage of gradient boosting to learn from weak learner trees.

4.1.4. Hyperparameter Tuning

For performance analysis, hyperparameter tuning was carried out on the validation data for the three model types.

1. Hyperparameters of the KNN model were tuned by testing odd values of k between 1 and 30, and it was found that k=1 produced the highest F1 score. The tuning was done for 2hrs.

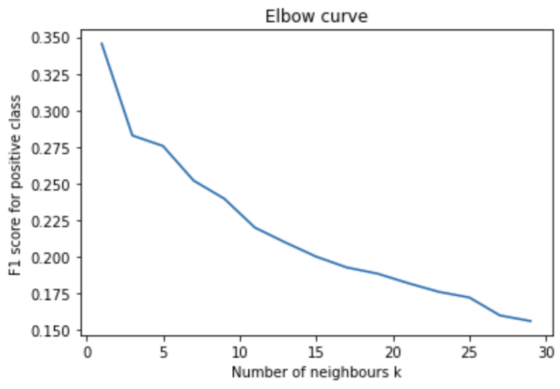
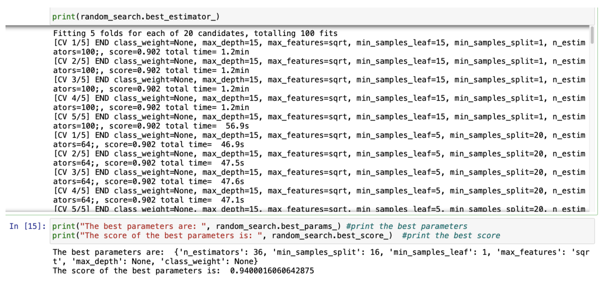


Fig.4.1.4.1 Tuning of k for KNN with F1 score

2. Hyperparameters of the Random Forest model were tuned by testing the different values for n_estimators, max_depth, max_features, min_samples_split, min_samples_leaf and class_weight. The tuning was done for 20 iterations with Randomsearch which took 27 hours.



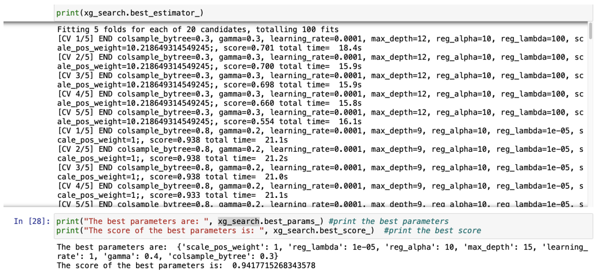
```
print(random_search.best_estimator_)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
[CV 1/5] END class_weight=None, max_depth=15, max_features=sqrt, min_samples_leaf=15, min_samples_split=1, n_estim
ators=100;, score=0.902 total time= 1.2min
[CV 2/5] END class_weight=None, max_depth=15, max_features=sqrt, min_samples_leaf=15, min_samples_split=1, n_estim
ators=100;, score=0.902 total time= 1.2min
[CV 3/5] END class_weight=None, max_depth=15, max_features=sqrt, min_samples_leaf=15, min_samples_split=1, n_estim
ators=100;, score=0.902 total time= 1.2min
[CV 4/5] END class_weight=None, max_depth=15, max_features=sqrt, min_samples_leaf=15, min_samples_split=1, n_estim
ators=100;, score=0.902 total time= 1.2min
[CV 5/5] END class_weight=None, max_depth=15, max_features=sqrt, min_samples_leaf=15, min_samples_split=1, n_estim
ators=100;, score=0.902 total time=  56.9s
[CV 1/5] END class_weight=None, max_depth=15, max_features=sqrt, min_samples_leaf=5, min_samples_split=20, n_estim
ators=64;, score=0.902 total time=  46.9s
[CV 2/5] END class_weight=None, max_depth=15, max_features=sqrt, min_samples_leaf=5, min_samples_split=20, n_estim
ators=64;, score=0.902 total time=  47.5s
[CV 3/5] END class_weight=None, max_depth=15, max_features=sqrt, min_samples_leaf=5, min_samples_split=20, n_estim
ators=64;, score=0.902 total time=  47.6s
[CV 4/5] END class_weight=None, max_depth=15, max_features=sqrt, min_samples_leaf=5, min_samples_split=20, n_estim
ators=64;, score=0.902 total time=  47.1s
[CV 5/5] END class_weight=None, max_depth=15, max_features=sqrt, min_samples_leaf=5, min_samples_split=20, n_estim
```

```
In [15]: print("The best parameters are: ", random_search.best_params_) #print the best parameters
         print("The score of the best parameters is: ", random_search.best_score_)  #print the best score

         The best parameters are:  {'n_estimators': 36, 'min_samples_split': 16, 'min_samples_leaf': 1, 'max_features': 'sqr
         t', 'max_depth': None, 'class_weight': None}
         The score of the best parameters is:  0.9400016060642875
```

Fig.4.1.4.2 Tuning of random forest with F1 score

3. Hyperparameters of the XGboost model were tuned by testing the different values for learning_rate, max_depth, gamma, colsample_bytree, reg_alpha, reg_lambda and scale_pos_weight (to account for class imbalance). The tuning was done for 20 iterations which took 3 hours.



```
print(xg_search.best_estimator_)

Fitting 5 folds for each of 20 candidates, totalling 100 fits
[CV 1/5] END colsample_bytree=0.3, gamma=0.3, learning_rate=0.0001, max_depth=12, reg_alpha=10, reg_lambda=100, sc
ale_pos_weight=10.218649314549245;, score=0.701 total time=  18.4s
[CV 2/5] END colsample_bytree=0.3, gamma=0.3, learning_rate=0.0001, max_depth=12, reg_alpha=10, reg_lambda=100, sc
ale_pos_weight=10.218649314549245;, score=0.700 total time=  15.9s
[CV 3/5] END colsample_bytree=0.3, gamma=0.3, learning_rate=0.0001, max_depth=12, reg_alpha=10, reg_lambda=100, sc
ale_pos_weight=10.218649314549245;, score=0.698 total time=  15.9s
[CV 4/5] END colsample_bytree=0.3, gamma=0.3, learning_rate=0.0001, max_depth=12, reg_alpha=10, reg_lambda=100, sc
ale_pos_weight=10.218649314549245;, score=0.660 total time=  15.8s
[CV 5/5] END colsample_bytree=0.3, gamma=0.3, learning_rate=0.0001, max_depth=12, reg_alpha=10, reg_lambda=100, sc
ale_pos_weight=10.218649314549245;, score=0.554 total time=  16.1s
[CV 1/5] END colsample_bytree=0.8, gamma=0.2, learning_rate=0.0001, max_depth=9, reg_alpha=10, reg_lambda=1e-05, s
cale_pos_weight=1;, score=0.938 total time=  21.1s
[CV 2/5] END colsample_bytree=0.8, gamma=0.2, learning_rate=0.0001, max_depth=9, reg_alpha=10, reg_lambda=1e-05, s
cale_pos_weight=1;, score=0.938 total time=  21.2s
[CV 3/5] END colsample_bytree=0.8, gamma=0.2, learning_rate=0.0001, max_depth=9, reg_alpha=10, reg_lambda=1e-05, s
cale_pos_weight=1;, score=0.938 total time=  21.0s
[CV 4/5] END colsample_bytree=0.8, gamma=0.2, learning_rate=0.0001, max_depth=9, reg_alpha=10, reg_lambda=1e-05, s
cale_pos_weight=1;, score=0.933 total time=  21.1s
[CV 5/5] END colsample_bytree=0.8, gamma=0.2, learning_rate=0.0001, max_depth=9, reg_alpha=10, reg_lambda=1e-05, s

In [28]: print("The best parameters are: ", xg_search.best_params_) #print the best parameters
         print("The score of the best parameters is: ", xg_search.best_score_)  #print the best score

The best parameters are:  {'scale_pos_weight': 1, 'reg_lambda': 1e-05, 'reg_alpha': 10, 'max_depth': 15, 'learning_
rate': 1, 'gamma': 0.4, 'colsample_bytree': 0.3}
The score of the best parameters is:  0.9417715268343578
```

Fig.4.1.4.3 Tuning of xgboost with F1 score

The above two hyperparameter is done using a cross validation of 5, which fits 5 model for a single model thus totalling 100 fits for a single model type.

4.1.5 Test Performance - Classification Report

The following are the classification report for the models.

```
              precision    recall  f1-score   support

           0       0.78      1.00      0.87     44591
           1       0.40      0.01      0.02     12645

    accuracy                           0.78     57236
   macro avg       0.59      0.50      0.45     57236
weighted avg       0.70      0.78      0.69     57236
```

Fig.4.1.5.1 KNN Classification Report

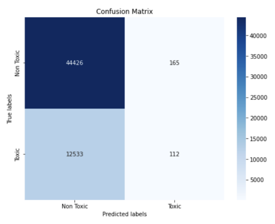
```
              precision    recall  f1-score   support

           0       0.80      1.00      0.89     44591
           1       0.88      0.11      0.20     12645

    accuracy                           0.80     57236
   macro avg       0.84      0.55      0.54     57236
weighted avg       0.82      0.80      0.73     57236
```

Fig.4.1.5.2 Random Forest Classification Report

```
              precision    recall  f1-score   support

           0       0.81      0.97      0.88     44591
           1       0.61      0.18      0.28     12645

    accuracy                           0.79     57236
   macro avg       0.71      0.57      0.58     57236
weighted avg       0.76      0.79      0.75     57236
```

Fig.4.1.5.3 Xgboost Classification Report

4.1.6 Test Performance - Confusion Matrix



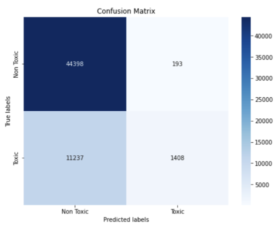


Fig.4.1.6.1. KNN   *Fig.4.1.6.2. Random Forest*

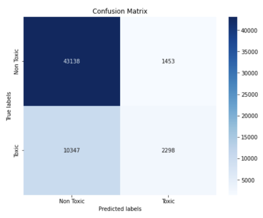


Fig.4.1.6.3. Xgboost

4.2 Approach 2: Transformers with feed forward

The proposed method would be a classification task that would identify toxic comments based on a multilingual dataset of comments. In the following approach, the toxic comments would be classified using a pre trained transformer architecture with customizations. Employing transformer architectures for classification in the field of NLP has been tried and tested in many past publications. One such example is the RoBERTa transformer. This model is based on the BERT transformer architecture with optimized hyperparameters and training. The RoBERTa transformer discards the classic new sentence pretraining objective of BERT. This means that RoBERTa efficiently creates mini-batches that facilitate quicker training with more context, rather than taking a sentence at a time. The RoBERTa dataset was also trained over a larger and wider dataset, hence being a good choice for transfer learning.

The transformer is tasked with extracting the required encoding matrix, while the feed forward network, characteristically implemented using Dense layers, trains over this matrix and provides the classification of toxicity. The binary cross-entropy function is used to evaluate the error. Due to the dataset distribution and to facilitate efficient training, pre-trained models have been used.

4.2.1 Training the Model

The model was trained for 7 epochs in a Keras Tensorflow environment using a Kaggle notebook. The training ran for over 2 hours and was evaluated on the test dataset using its performance metrics (accuracy, precision, recall, f1-score)

4.2.2 Results

The results for the model can be found below. The results include a classification report and a confusion matrix.

```
CONFUSION MATRIX
```

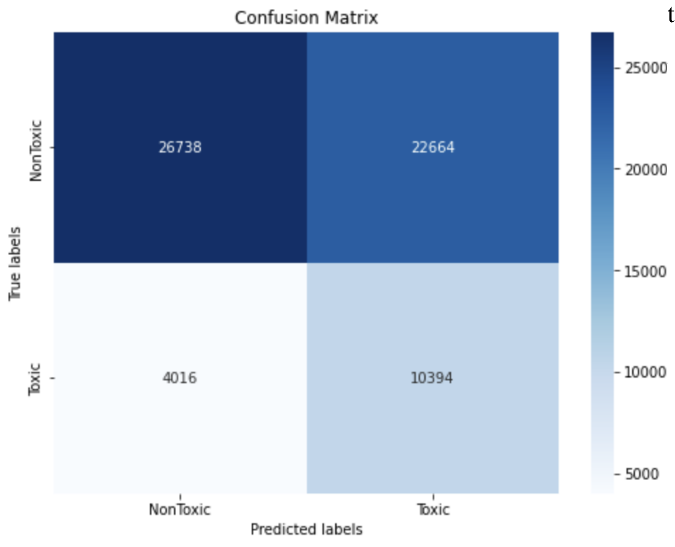
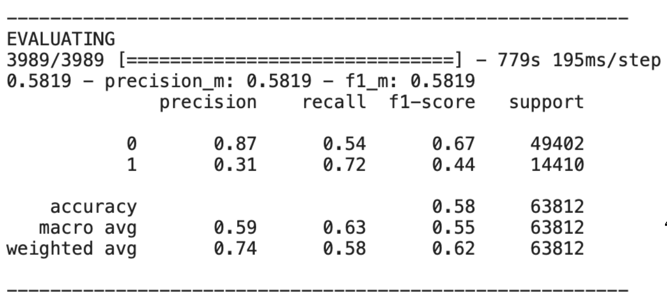


Fig 4.2.2.1: Confusion Matrix for Transformer + FFNN



```
__________________________________________________________
EVALUATING
3989/3989 [==============================] - 779s 195ms/step
0.5819 - precision_m: 0.5819 - f1_m: 0.5819
              precision    recall  f1-score   support

           0       0.87      0.54      0.67     49402
           1       0.31      0.72      0.44     14410

    accuracy                           0.58     63812
   macro avg       0.59      0.63      0.55     63812
weighted avg       0.74      0.58      0.62     63812

__________________________________________________________
```

Fig 4.2.2.2: Classification Report for Transformer + FFNN

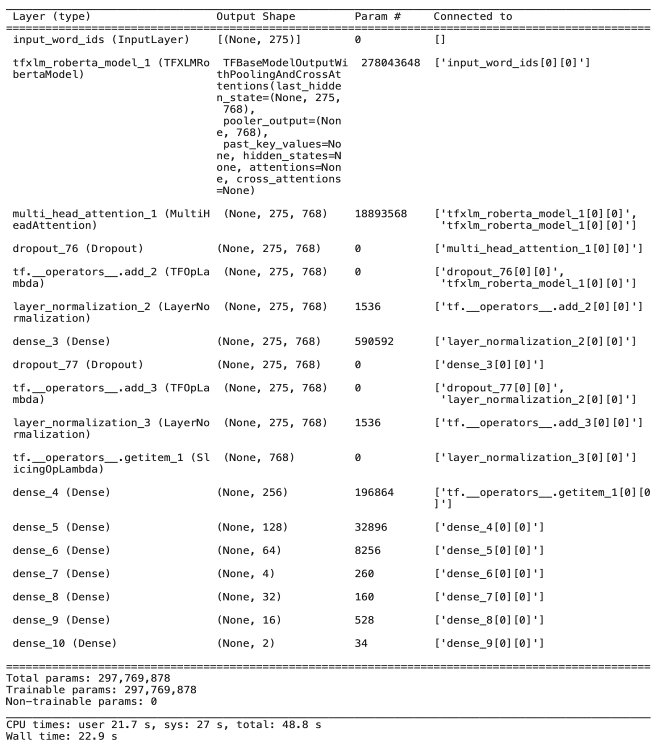
4.3 Approach 3: Stacked transformers

Since we are using a GPT-2 based Transformer model (XLM-RoBERTa-base) and adding a feed forward layer at the end. We want to experiment on adding a Multi-headed Attention Transformer block stacked on top of the Base RoBERTa model. We got the idea from Inception based transfer learning architecture, where adding an inception block after the Base Inception network increases the performance of the model and allows the model to learn complex features.

We experimented the above ideology in our experiment by adding a Vanilla Transformer block from after the RoBERTa model followed by Encoder-Decoder style feed forward network with softmax activation function for classification of Toxic and Non-Toxic Comments. We trained this architecture on both sampled and un-sampled data as a part of our project.

4.3.1 Model Building

To build the model we utilized keras and Hugging Face libraries in python to architect the model. The model is designed with the XLM-RoBERTa model as a feature encoder followed by a custom Transformer block with *num_heads* as 8, *feed_forward_dim* as 768, and dropout connections between them. The transformer model architecture is given below.

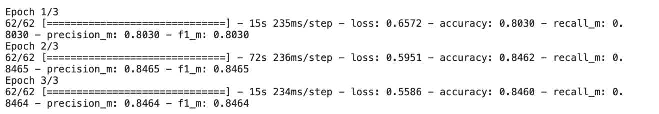


```
____________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
============================================================================================
 input_word_ids (InputLayer)    [(None, 275)]        0           []

 tfxlm_roberta_model_1 (TFXLMRo  TFBaseModelOutputWi  278043648  ['input_word_ids[0][0]']
 bertaModel)                    thPoolingAndCrossAt
                                tentions(last_hidde
                                n_state=(None, 275,
                                 768),
                                 pooler_output=(Non
                                e, 768),
                                 past_key_values=No
                                ne, hidden_states=N
                                one, attentions=Non
                                e, cross_attentions
                                =None)

 multi_head_attention_1 (MultiH  (None, 275, 768)    18893568    ['tfxlm_roberta_model_1[0][0]',
 eadAttention)                                                    'tfxlm_roberta_model_1[0][0]']

 dropout_76 (Dropout)           (None, 275, 768)     0           ['multi_head_attention_1[0][0]']

 tf.__operators__.add_2 (TFOpLa  (None, 275, 768)    0           ['dropout_76[0][0]',
 mbda)                                                            'tfxlm_roberta_model_1[0][0]']

 layer_normalization_2 (LayerNo  (None, 275, 768)    1536        ['tf.__operators__.add_2[0][0]']
 rmalization)

 dense_3 (Dense)                (None, 275, 768)     590592      ['layer_normalization_2[0][0]']

 dropout_77 (Dropout)           (None, 275, 768)     0           ['dense_3[0][0]']

 tf.__operators__.add_3 (TFOpLa  (None, 275, 768)    0           ['dropout_77[0][0]',
 mbda)                                                            'layer_normalization_2[0][0]']

 layer_normalization_3 (LayerNo  (None, 275, 768)    1536        ['tf.__operators__.add_3[0][0]']
 rmalization)

 tf.__operators__.getitem_1 (Sl  (None, 768)         0           ['layer_normalization_3[0][0]']
 icingOpLambda)

 dense_4 (Dense)                (None, 256)          196864      ['tf.__operators__.getitem_1[0][0
                                                                 ]']

 dense_5 (Dense)                (None, 128)          32896       ['dense_4[0][0]']

 dense_6 (Dense)                (None, 64)           8256        ['dense_5[0][0]']

 dense_7 (Dense)                (None, 4)            260         ['dense_6[0][0]']

 dense_8 (Dense)                (None, 32)           160         ['dense_7[0][0]']

 dense_9 (Dense)                (None, 16)           528         ['dense_8[0][0]']

 dense_10 (Dense)               (None, 2)            34          ['dense_9[0][0]']

============================================================================================
Total params: 297,769,878
Trainable params: 297,769,878
Non-trainable params: 0
____________________________________________________________________________________________
CPU times: user 21.7 s, sys: 27 s, total: 48.8 s
Wall time: 22.9 s
```

Fig 4.3.1.1: Stacked Transformer Model Architecture

4.3.2 Training the Model

Once the architecture has been decided, we trained the model using TPU's provided by Kaggle. By iterating through various versions and architecture, we decided to go with the above architecture yielding better results. This model has been trained in the TPU for over 3 hours. We couldn't experiment and fine-tune the hyperparameters due to limitations in computational resources. Once the training has been done on the training dataset, we trained on the validation dataset too. This is done to capture various distributions of the dataset and to make out model more robust.



```
Epoch 1/3
62/62 [==============================] - 15s 235ms/step - loss: 0.6572 - accuracy: 0.8030 - recall_m: 0.
8030 - precision_m: 0.8030 - f1_m: 0.8030
Epoch 2/3
62/62 [==============================] - 72s 236ms/step - loss: 0.5951 - accuracy: 0.8462 - recall_m: 0.
8465 - precision_m: 0.8465 - f1_m: 0.8465
Epoch 3/3
62/62 [==============================] - 15s 234ms/step - loss: 0.5586 - accuracy: 0.8460 - recall_m: 0.
8464 - precision_m: 0.8464 - f1_m: 0.8464
```

Fig 4.3.2.1: Training Metrics on validation data

4.3.3 Test Performance: Classification Report

The following are the classification reports for the models. As illustrated by the metrics, stacking a transformer on top of the BERT architecture has skewed the prediction results. There is a difference in the theoretical hypothesis and the actual results. One possible reason might be the very low training time and epochs, this is unavoidable as we lack the technical resources to train the Transformer model over TPU for extended time. Robust hyper-parameter tuning will also help the model.

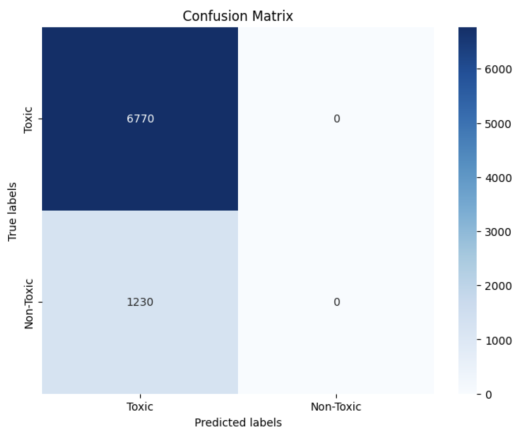
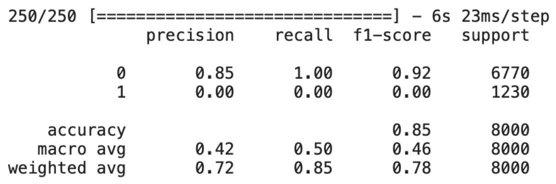


Fig 4.3.3.1: Confusion Matrix for Stacked Transformer



```
250/250 [==============================] - 6s 23ms/step
              precision    recall  f1-score   support

           0       0.85      1.00      0.92      6770
           1       0.00      0.00      0.00      1230

    accuracy                           0.85      8000
   macro avg       0.42      0.50      0.46      8000
weighted avg       0.72      0.85      0.78      8000
```

Fig 4.3.3.2: Classification Report for Stacked Transformer

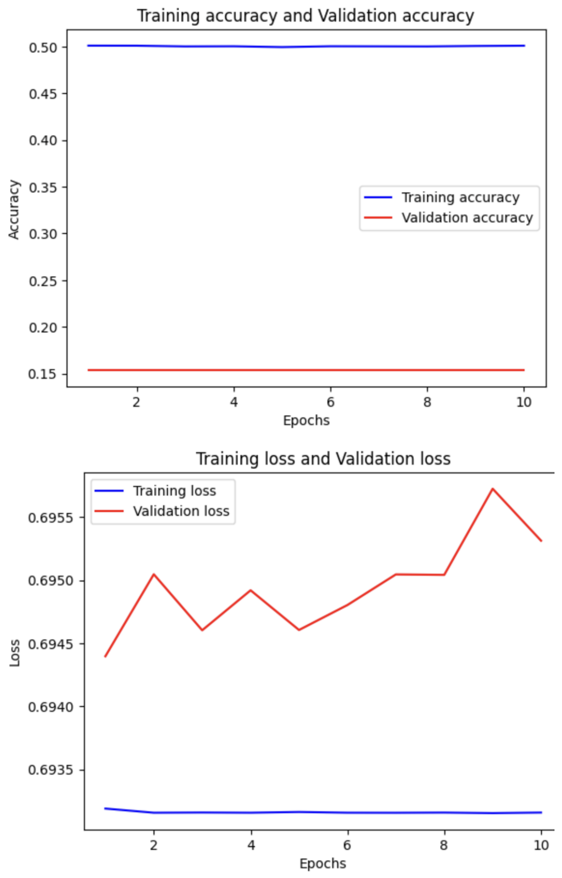


Fig 4.3.3.3: Training Graphs for Stacked Transformer

Algorithm	**Hypothesis**	**Approach**	**Cost Function**	**Hyper Param.**
KNN	Majority vote of its k neighbors	Model is majority vote of its k-neighbors	No cost function	K
Random Forest	Combines the prediction of multiple decision trees	Each trees are trained on a random subset of the training data, robustness against overfitting.	Cross Entropy	n_estimators, max_depth, max_features, min_samples_split, min_samples_leaf, class_weight
XGBoost	Uses Gradient Boosting technique to learn from weak learners	The algorithm works by creating an ensemble of decision trees and adjusting the weights of the different trees to predict.	Cross Entropy	learning_rate, max_depth, gamma, colsample_bytree, reg_alpha, reg_lambda, scale_pos_weight
RoBERTa - FFNN	Employs a BERT like architecture with optimized hyperparameters	The algorithm discards the NSP objective and trains on longer batches of sentences. Also employs a dynamic	Cross Entropy	Learning_rate, max_len

		masking technique. Adds Dense layers to facilitate a FFNN.		
RoBERTa - Stacked	Employs a BERT like architecture with a vanilla transformer stacked on top of it	This algorithm is trained on longer batched of sentences. Also employs a dynamic masking technique. Adds a Vanilla Transformer block at the end with BottleNeck layer based feed forward network for classification	Cross Entropy	Learning_rate, chunk_size, max_len, num_heads, feed_forward_dim, dropout percentage

Table.1 Summary of the mathematical background of classification models

5. Experiment

5.1 Data Description

The dataset employed in this project consists of multilingual comments that can be classified into 5 classes of increasing levels of toxicity [7]. The open-source dataset is designed in such a way that the toxicity of a comment can be predicted as a probability from 0 to 1. There are 223,549 samples in the Train set with 63,812 samples in the Test set. Each sample originally had a respective rating of toxicity across 6 different labels of toxicity: toxic, severe toxic, obscene, threat, insult, and identity hate.

In order to facilitate the objective, the dataset employed in this study was preprocessed to categorize binary toxicity and non-toxicity. A train test split ratio of 70:30 was employed.

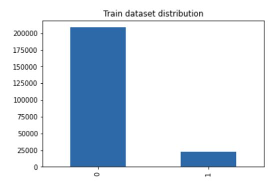


Fig 5.1.1. Train

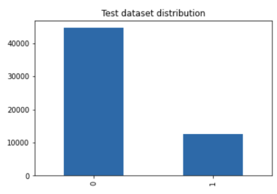


Fig 5.1.2. Test

5.2 Experimental Results

The performance of different learning algorithms on the same dataset is tested and evaluated over their f1-score.

5.2.1 Performance Measurement

Here, the proposed supervised algorithms are trained for the same toxic comments dataset and their predictions for the test data were measured. Figure 5 demonstrates a performance comparison across the different models tested.

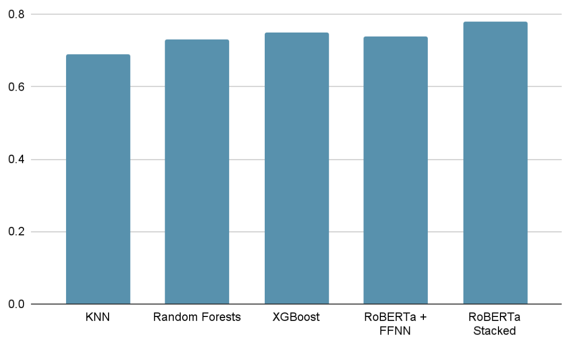


Figure.5 Prediction F1-Score of classification models

It is shown in Figure 5 that the use of machine learning algorithms like KNN, Random Forest, and XGBoost are comparable to transformers in the overview. For the given dataset, however, there is a slight boost in the f1-score reported by the transformer models comparatively (Stacked transformer >=78%). The better performance of these transformers can also be seen from their individual classification reports and confusion matrices. It is visible in both the ML and transformer approaches that the performance of the models on the toxic class-1 label is very low. These results were obtained on the test dataset. There is a comparatively balanced performance reported by the FFNN transformer instead of stacking. While stacking reports higher f1-score, there is a higher rate of misclassification. This could be due to the fact that the features extracted by the transformer are complex, and the further transformer operations on these features result in the loss of contextual info. The neural network fares well in this regard.

Apart from the f1-score, the running time of these models are a significant factor to consider. Transformers are characteristically time and computationally consuming. The run time logged by our models can be seen in table 2.

Classification Model	Time (min)
K Nearest Neighbors	8.8
Random Forest	73
XGBoost	2.5
RoBERTa with FFNN	120
Stacked transformer	100

Table.2 Running Time for training and testing of the proposed models

6.Summary and Discussion:

Classification ML algorithms are fundamentally strong in their ability to identify generalized patterns amongst simple datasets. However, they lack the ability to work with text using the context of a sentence.

On the other hand, transformers have been trending recently for their ability to identify temporal characteristics and maintain a generalized detection at the same time. However, transformers require high-computational resources and often need large dimensional data for verifiable results.

In this study, we compare the use of both these methods and analyze them based on their performance. From the experimental results detailed in section 5.2 it can be seen that transformers perform better in terms of identifying the contextual differences between toxic and non toxic data. However, transformers are time consuming and computationally expensive, which was an underlying limitation of this study as well. Nevertheless, the use of optimized machine learning models have also been proven to be promising. But at the end, the combination of feature extraction through transformers with a feed forward neural network classifier provides the best results in this study.

In future work, we expect to expand into a larger dataset, with better computational resources. While the effectiveness of using transformers as a feature extractor has been verified, the development of a unique classification head can be undertaken for this theme of toxic comment classification.

7. References:

1. https://www.omnicoreagency.com/twitter-statistics/
2. Zaheri, Sara; Leath, Jeff; and Stroud, David (2020) "Toxic Comment Classification," SMU Data Science Review: Vol. 3: No. 1, Article 13. Available at: https://scholar.smu.edu/datasciencereview/vol3/iss1/1
3. A. Garlapati, N. Malisetty and G. Narayanan, "Classification of Toxicity in Comments using NLP and LSTM," 2022 8th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, 2022, pp. 16-21, doi: 10.1109/ICACCS54159.2022.9785067.
4. Pavel, M. I., Razzak, R., Sengupta, K., Niloy, M. D. K., Muqith, M. B., & Tan, S. Y. (2021). Toxic comment classification implementing CNN combining word embedding technique. In Inventive Computation and Information Technologies: Proceedings of ICICIT 2020 (pp. 897-909). Springer Singapore. Chicago
5. Zhixue Zhao, Ziqi Zhang, and Frank Hopfgartner. 2021. A Comparative Study of Using Pre-trained Language Models for Toxic Comment Classification. In Companion Proceedings of the Web Conference 2021 (WWW '21). Association for Computing Machinery, New York, NY, USA, 500–507. https://doi.org/10.1145/3442442.3452313
6. Akash, G., Kumar, H., & Bharathi, D. (2021). Toxic comment classification using transformers. In Proceedings of the 11 th Annual International Conference on Industrial Engineering and Operations Management Singapore (pp. 1895-1905).
7. **DATA SOURCE**: https://www.kaggle.com/competitions/jigsaw-multilingual-toxic-comment-classification/data