CyberGuard
Detection of inappropiate content on Social Media indicating Cyberbullying

22114058 **Aditya Mundada**
22114060 **Nayan Kakade**
22114092 **Shubham Kumar Verma**

Submitted in partial fulfillment of the requirements for the Degree of
BACHELOR IN TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING

Supervisor : Prof.Pravendra Singh

Contents

Chapter 1

Members and their contributions

Aditya Mundada: 22114058

- Developed machine learning models, including Logistic Regression, Support Vector Machine (SVM), and Random Forest and LSTM.
- Conducted hyperparameter tuning and model evaluation using metrics such as accuracy, precision, recall, and F1-score.
- Compared performance of various models drawing conclusions and authored the final report.

Nayan Kakade : 22114060

- Performed initial data preprocessing, including text cleaning, tokenization, and stopword removal.
- Implemented feature extraction techniques such as TF-IDF and word embeddings.
- Research on LSTM development and its deployment.

Shubham Verma: 22114092

- Created a balanced dataset by combining multiple datasets and helped in data cleansing.
- Assisted in exploratory data analysis (EDA) and data visualization.
- Performed BERT fine tuning and research on its significance.

Chapter 2

Motivation

The rise of the 21st century has seen civilisation transform into the digital era with social media platforms serving as vital communication channels where people express their opinions, grievances and keep others informed regarding their day to day lives.
However, these platforms have also become hotspots for cyberbullying. Cyberbullying includes instances of trolling and abusing people online through mean comments, messages, images with the sole aim of defamation of the victim.
Different mechansims exist for identifying instances of cyberbullying, the most common being using peer review, feedback and report system to flag bullies. Though, this system works well, it is not scalable since it misses automation and using it against bots created exclusively for bullying purposes is infeasible.
Cyberguard focuses on developing an automated system to detect inappropriate content indicative of cyberbullying across various online platforms. We aim to use NLP techniques and different ML algorithms to accurately detect instances of cyberbullying on the internet which may involve abuse, attack, aggression or sexist and racist abuses. We also aim to work in a more Indian context with identifying trolls in languages like Hinglish too.

Chapter 3

Dataset

3.1 Dataset Description

We have combined 3 datasets to create our own dataset that comprises various features we required.

- We first used the cyberbullying_tweets.csv which includes about **47000** tweet instances from various categories of cyberbulling like age, ethnicity, gender as well as some non-offensive examples.
- We also have the Hinglish dataset which has a few offensive and non-offensive Hinglish and English Examples.
- The dataset created by merging these two datasets is highly imbalanced and can hamper the training of the models. In order to create balance and increase the weightage of non-offensive tweets, we randomly chose non-offensive tweets from the Aggression dataset. This balanced our dataset to include an approximately equal ratio of offensive and non-offensive training samples.

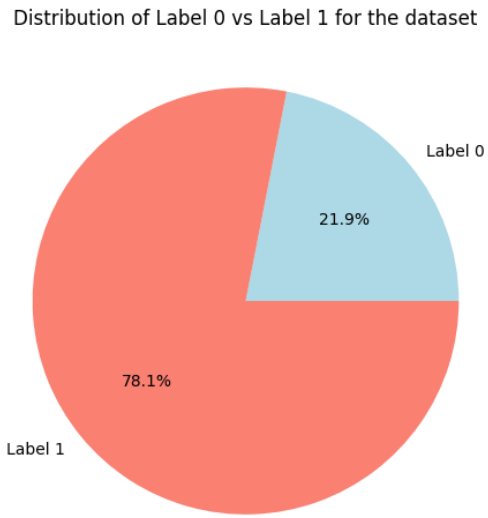


Figure 3.1: Imbalanced dataset distribution before we added instances from the aggression dataset

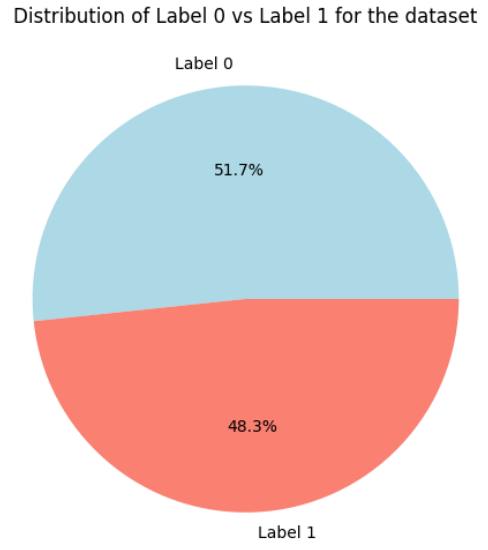


Figure 3.2: Balanced dataset distribution after adding instances from the aggression dataset

3.2 Data Cleansing and Preprocessing

To prepare the raw textual data for effective feature extraction, we implemented a comprehensive preprocessing pipeline. This stage is critical for reducing noise and standardizing the input text prior to model training. The key steps in our preprocessing include:

- **URL Removal:** Eliminating all web links using regular expressions to discard extraneous information.
- **Special Character and Emoji Elimination:** Removing punctuation, non-alphanumeric characters and emojis to reduce textual noise.
- **Case Normalization:** Converting all text to lowercase, which minimizes the vocabulary size by treating similar words uniformly.
- **Tokenization:** Splitting text into individual tokens, thereby facilitating subsequent analysis.
- **Stopword Removal and Lemmatization:** Filtering out common stopwords and lemmatizing tokens to convert words to their base forms, thus enhancing the quality of the features extracted. We have also included the stopwords from hindi to support Hinglish text.

```
def clean_tweet(tweet):
    tweet = remove_emoji(tweet)
    tweet = expand_contractions(tweet)
    tweet = remove_all_entities(tweet)
    tweet = clean_hashtags(tweet)
    tweet = remove_chars(tweet)
    tweet = remove_mult_spaces(tweet)
    tweet = remove_numbers(tweet)
    tweet = lemmatize(tweet)
    tweet = remove_short_words(tweet)
    tweet = correct_elongated_words(tweet)
    tweet = remove_repeated_punctuation(tweet)
    tweet = remove_extra_whitespace(tweet)
    tweet = remove_url_shorteners(tweet)
    tweet = remove_spaces_tweets(tweet)
    tweet = remove_short_tweets(tweet)
    tweet = '␣'.join(tweet.split())
    return tweet
```

3.3 Exploratory dataset analysis of merged dataset

The merged dataset which we have used for training includes **93552** samples. After cleansing it has only two columns, cleaned text and label (0/1). 0 indicates absence of bullying

while 1 indicates presence.

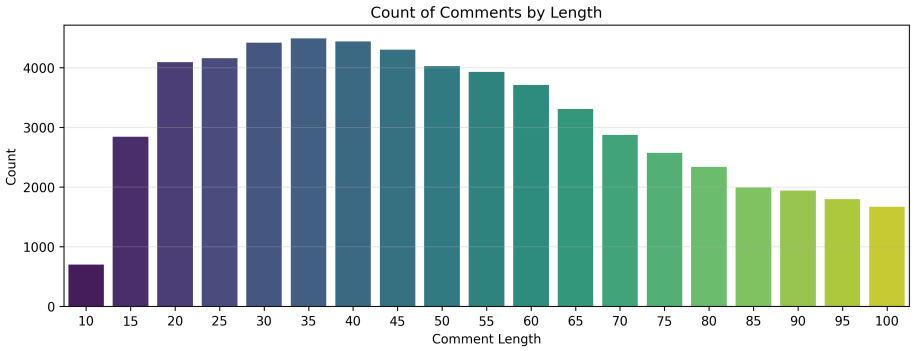


Figure 3.3: Distribution of comment length

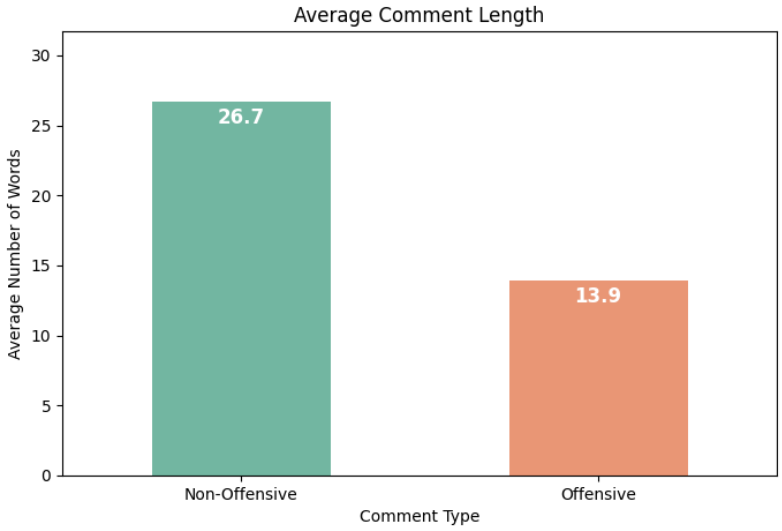


Figure 3.4: Average comment length for offensive and non-offensive texts

We can clearly observe that offensive comments are shorter than non-offensive ones.

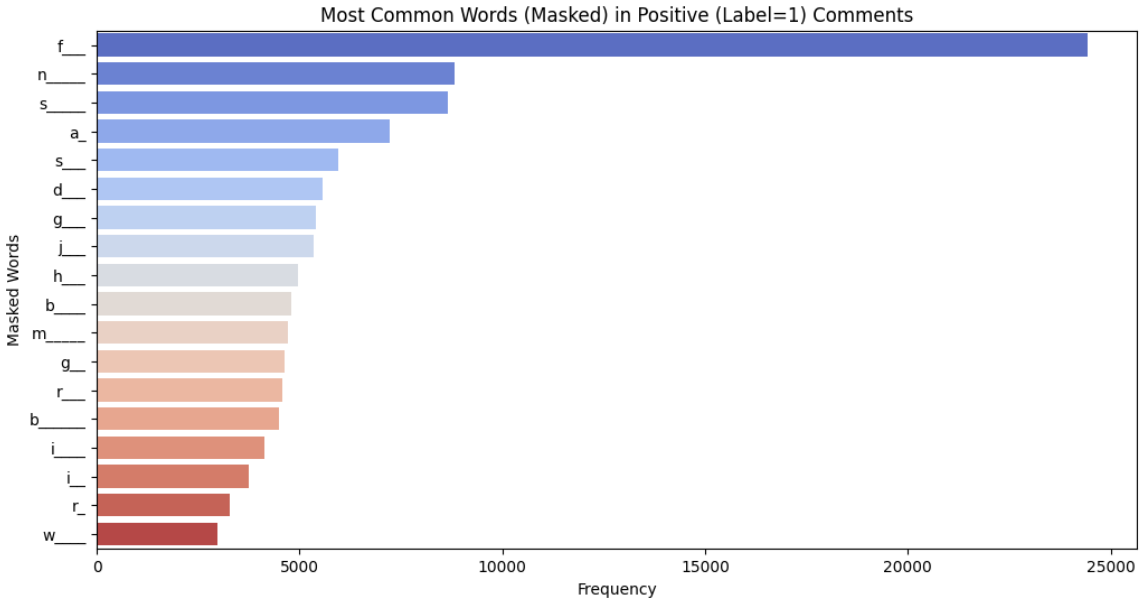


Figure 3.5: Most common abusive words(Masked)

Chapter 4

Traditional Models Used

Before, applying traditional models, we must perform vectorisation.

4.1 Vectorisation

It is the process of converting data into numerical representation. Algorithms need input in the form of numbers to operate upon them.

4.1.1 TF-IDF (Term Frequency - Inverse Document Frequency)

Reflects the importance of a word in a document relative to a corpus.

Term Frequency:
The number of times a word appears in a document

$$TF(w) = \frac{\text{Number of times word } w \text{ appears in a document}}{\text{Total words in document}}$$

Inverse Document Frequency:
A measure of how many other documents in a corpus contain the same word

$$IDF(w) = \log\left(\frac{\text{Total words in documents}}{\text{Number of documents containing } w}\right)$$

TF-IDF score

$$TF\_IDF(w) = TF(w) \times IDF(w)$$

4.1.2 Example

Tokenized Text = ['hate', 'you', 'stupid']

Word	TF	IDF	TF-IDF Score
hate	0.2	1.5	0.3
you	0.1	0.5	0.05
stupid	0.15	1.7	0.255

Vector after vectorization = `[0.3, 0.05, 0.255]`

4.2 Models used

In this study, we explore three widely used traditional machine learning models for cyberbullying detection:

4.2.1 Logistic Regression

Logistic Regression is a supervised learning algorithm used for classification tasks. The output is a probability value between 0 and 1, making it useful for classification problems. The logistic (sigmoid) function is defined as:

$$f(z) = \frac{1}{1 + e^{-z}}, \quad 0 \leq f(z) \leq 1 \tag{4.1}$$

where z is a linear combination of independent variables:

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k \tag{4.2}$$

Here,

- β_0 is the **intercept**.
- $\beta_1, \beta_2, \ldots, \beta_k$ are the **slopes** (coefficients) corresponding to independent variables $x_1, x_2, \ldots, x_k$.

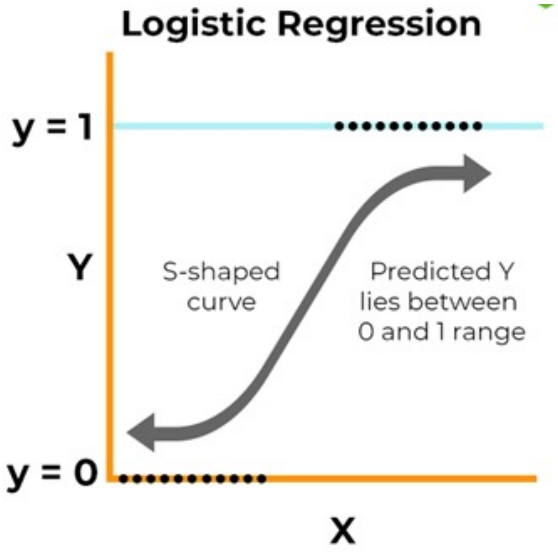


Figure 4.1: S-curve of logistic function

The logistic function maps the linear combination z onto the range $[0, 1]$, producing an S-shaped curve. The predicted probability helps in decision-making for classification problems.
In order to extend Logistic regression to multiclass classification problems: There are two common approaches to solving multiclass classification problems:

- **One-vs-Rest (OvR) Binary Classification:** This approach involves training N separate logistic regression models, where N is the number of classes. Each model distinguishes one class from all others.
- **Softmax Function:** Instead of training multiple models, the softmax function assigns a probability to each class. The probability of a given instance X belonging to class j is computed as:

$$P(y = j|X) = \frac{e^{\beta_j^T X}}{\sum_{k=1}^{N} e^{\beta_k^T X}} \tag{4.3}$$

  where:

  - β_j is the coefficient vector for class j,
  - X is the feature vector,
  - N is the total number of classes.

4.2.2 Random Forests

Random Forests is an ensemble learning method that builds multiple decision trees and aggregates their predictions for better generalization.
They use bootstrap aggregating and random feature selection for robustness.

For classification, the final prediction is determined by majority voting:

$$\hat{y} = \arg\max_{c} \sum_{m=1}^{M} 1(f_m(x) = c) \tag{4.4}$$

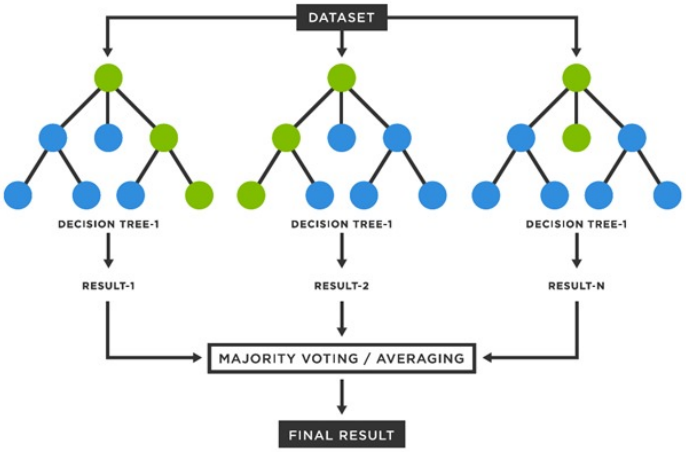


Figure 4.2: Random forests as ensemble methods

Random Forests reduce overfitting and improve model stability by leveraging multiple weak learners.

4.2.3 Support Vector Classifier (SVC)

Support Vector Classification (SVC) is a supervised learning algorithm used classification. It aims to find the optimal hyperplane that maximizes the margin between different classes.

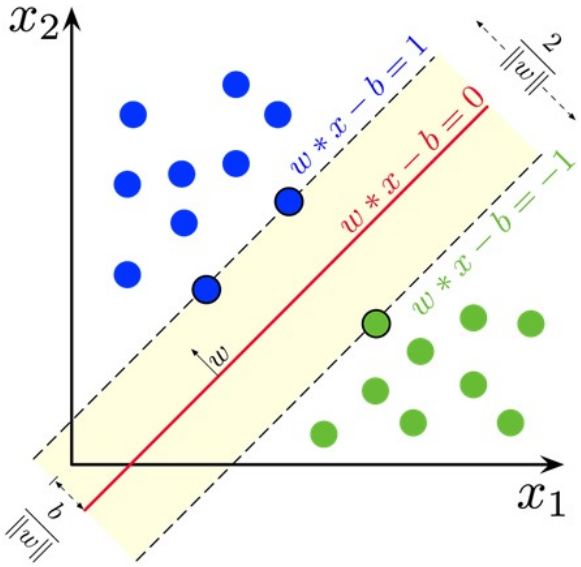


Figure 4.3: Support vector classifiers

Given a training set (x_i, y_i) where x_i are feature vectors and $y_i \in \{-1, 1\}$ are class labels, SVC solves the following optimization problem:

$$\min_{\beta, b} \frac{1}{2}\|\beta\|^2 \tag{4.5}$$

subject to:

$$y_i(\beta^T x_i + b) \geq 1, \quad \forall i \tag{4.6}$$

where:

- β is the weight vector defining the hyperplane,
- b is the bias term,
- The constraint ensures that all data points are correctly classified with a margin of at least 1.

Chapter 5

Simple LSTM

5.1 What is an LSTM?

Long Short-Term Memory (LSTM) networks are a type of Recurrent Neural Network (RNN) capable of learning long-term dependencies. They are explicitly designed to avoid the long-term dependency problem, which traditional RNNs struggle with due to vanishing gradients.
An LSTM cell maintains a cell state that runs through the entire sequence chain with only minor linear interactions. It includes three gates:

- **Forget gate:** Decides what information to throw away from the cell state.
- **Input gate:** Updates the cell state with new information.
- **Output gate:** Decides what to output based on the cell state.

This architecture allows LSTMs to remember patterns over long sequences, making them suitable for tasks like language modeling and sentiment classification.

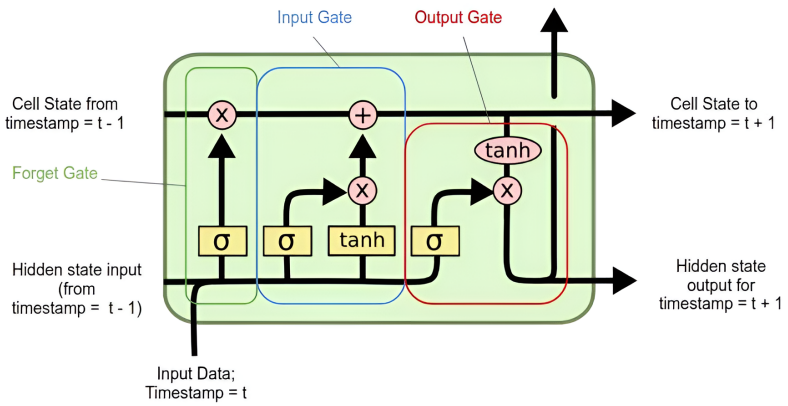


Figure 5.1: LSTM structure

5.2 Model Architecture

The implemented LSTM model uses the following layers:

1. **Embedding Layer:**

   $$\text{Embedding(input\_dim=V, output\_dim=200, input\_length=100)}$$

   where V is the vocabulary size. This layer converts word indices into dense vectors of size 200.

2. **LSTM Layer:**

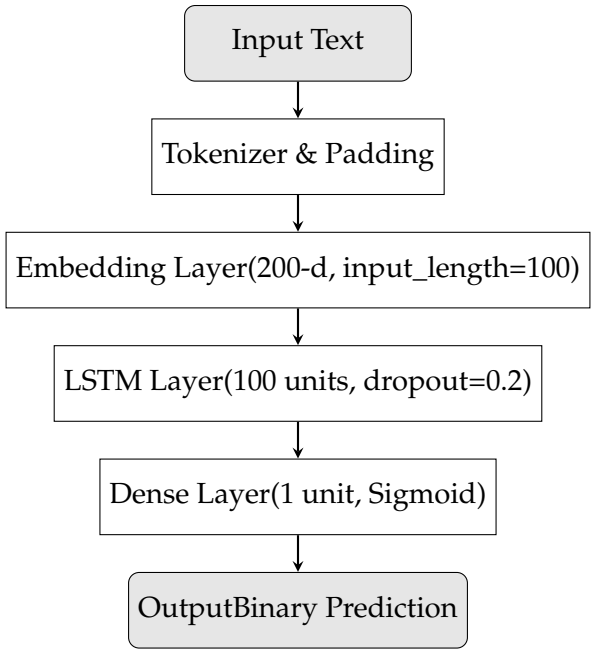
   $$\text{LSTM(units=100, dropout=0.2, recurrent\_dropout=0.2)}$$

   This layer contains 100 LSTM units that process the sequence and capture dependencies.

3. **Dense Output Layer:**

   $$\text{Dense(units=1, activation='sigmoid')}$$

   Produces a single output probability for binary classification.



5.3 Loss Function

The model uses Binary Cross-Entropy as the loss function:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

where:

- y_i is the true label
- $\hat{y}_i$ is the predicted probability
- N is the number of samples

5.4 Optimizer: Adam

The Adam optimizer is used for training, which combines ideas from RMSProp and Momentum. The update rule is given by:

$$\theta_t = \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

where:

- $\hat{m}_t$ is the bias-corrected first moment (mean of gradients)
- $\hat{v}_t$ is the bias-corrected second moment (uncentered variance)
- α is the learning rate

5.5 Training Process

The model is trained using mini-batch gradient descent with a batch size of 32 and trained for up to 10 epochs. Early stopping is used to monitor validation loss and avoid overfitting. The best model (based on validation loss) is saved separately.

5.6 Loss curve

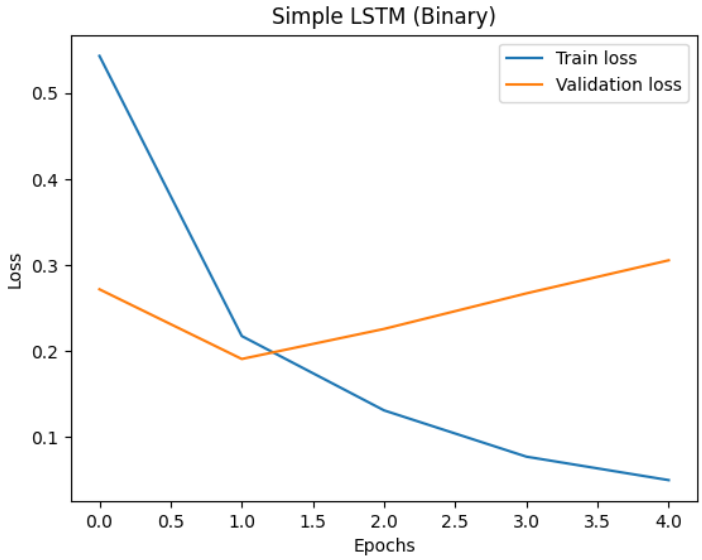


Figure 5.2: Loss curve for LSTM

Chapter 6

BERT

6.1 What is BERT?

Bidirectional Encoder Representations from Transformers (BERT) is a transformer-based model introduced by Google in 2018. It is designed to pre-train deep bidirectional representations by jointly conditioning on both left and right context in all layers. Unlike traditional models, which read text input sequentially (left-to-right or right-to-left), BERT reads entire sequences of words at once, allowing it to develop a deeper understanding of context. We have used BERT and fine-tuned it on our dataset for cyberbullying detection.

6.2 BERT architecture

BERT's architecture is based on the Transformer encoder. The base model consists of:

- 12 Transformer blocks
- 768 hidden size
- 12 self-attention heads
- 110 million parameters

It is trained using two objectives:

1. **Masked Language Modeling (MLM)** – Random words are masked and the model learns to predict them.
2. **Next Sentence Prediction (NSP)** – The model learns the relationship between two sentences.

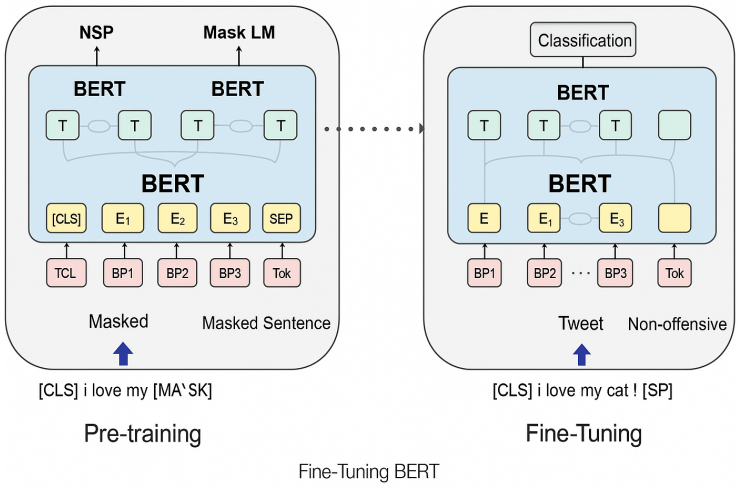


Figure 6.1: Fine-tuning Bert

6.3 BERT for cyberbullying detection

The following steps were implemented to train BERT for binary classification:

1. **Data Preparation:** The dataset was split into training and validation sets using `train_test_split`. The cleaned text and labels were extracted and converted into lists.

2. **Tokenization:** The `BertTokenizer` from the `bert-base-uncased` model was used to tokenize the text, with padding and truncation applied for a consistent sequence length of 128.

3. **Dataset Class:** A custom PyTorch Dataset class was created, named `CyberBullyingDataset`, to structure the inputs and labels for DataLoader compatibility.

4. **DataLoader:** PyTorch `DataLoaders` were created for both training and validation sets with a batch size of 16.

5. **Model Loading:** The `BertForSequenceClassification` model was loaded with 2 output labels for binary classification.

6. **Optimizer Setup:** The AdamW optimizer was initialized with a learning rate of 5×10^{-5}.

7. **Training Loop:** The model was trained for 3 epochs. For each batch, forward and backward propagation was performed, and the optimizer updated the model weights accordingly.

6.4 Optimiser: AdamW

AdamW is an adaptive optimization algorithm that combines the benefits of Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp), while correctly decoupling weight decay.
Let:

- θ_t be the parameters at time step t
- $g_t = \nabla_\theta L(\theta_t)$ be the gradient of the loss function at time step t
- m_t be the first moment estimate (mean of gradients)
- v_t be the second moment estimate (uncentered variance of gradients)
- β_1, β_2 be the exponential decay rates for the moment estimates
- ϵ be a small constant to prevent division by zero
- η be the learning rate
- λ be the weight decay coefficient

The AdamW update rules are:

$$
\begin{aligned}
m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\
v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \\
\hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\
\hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \\
\theta_{t+1} &= \theta_t - \eta \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda \theta_t \right)
\end{aligned}
$$

The key difference from the standard Adam optimizer lies in the last update step, where $\lambda\theta_t$ is explicitly added, decoupling the weight decay from the gradient-based update.

Chapter 7

Evaluation Criteria

The following metrics were computed to measure model performance:

- **Accuracy**: The proportion of correctly predicted instances out of the total instances.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{7.1}$$

- **Precision**: The ratio of correctly identified cyberbullying cases (true positives) to the total predicted cases (true positives + false positives), indicating the model's exactness.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{7.2}$$

- **Recall**: The ratio of true positives to the sum of true positives and false negatives, measuring the model's ability to capture all relevant instances.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{7.3}$$

- **F1-Score**: The harmonic mean of precision and recall, providing a balanced metric.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{7.4}$$

Where:

- TP = True Positives
- TN = True Negatives
- FP = False Positives
- FN = False Negatives

Chapter 8

Model Performance summary

8.1 Logistic Regression

Class	Precision	Recall	F1-score
Absent	0.82	0.93	0.87
Present	0.91	0.78	0.84
Macro Avg	0.86	0.85	0.85
Weighted Avg	0.86	0.86	0.85

Table 8.1: Logistic Regression: **Accuracy=0.86**

8.2 Random Forest classifier

Class	Precision	Recall	F1-score
Absent	0.82	0.94	0.88
Present	0.92	0.78	0.85
Macro Avg	0.87	0.86	0.86
Weighted Avg	0.87	0.86	0.86

Table 8.2: Random Forest: **Accuracy=0.86**

8.3 Support vector classifier

Class	Precision	Recall	F1-score
Absent	0.82	0.94	0.87
Present	0.92	0.78	0.84
Macro Avg	0.87	0.86	0.86
Weighted Avg	0.87	0.86	0.86

Table 8.3: SVC: **Accuracy=0.86**

8.4 LSTM

Class	Precision	Recall	F1-score
Absent	0.91	0.93	0.92
Present	0.92	0.90	0.91
Macro Avg	0.92	0.92	0.92
Weighted Avg	0.92	0.92	0.92

Table 8.4: LSTM: **Accuracy=0.92**

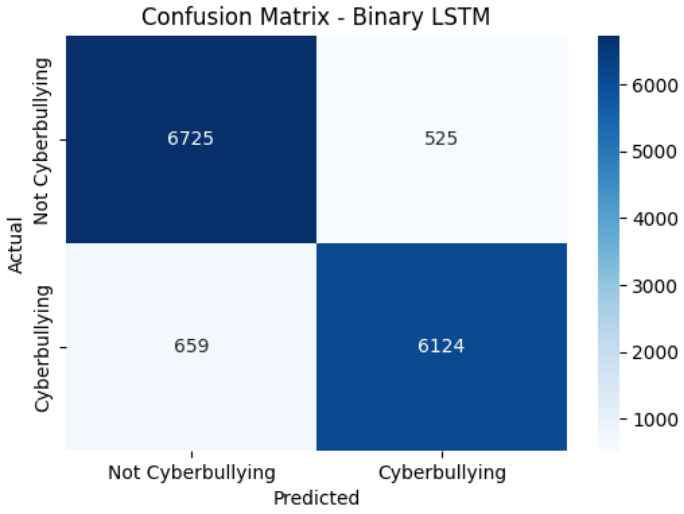


Figure 8.1: Confusion Matrix for LSTM

8.5 Fine-tuned BERT

Class	Precision	Recall	F1-score
Absent	0.90	0.94	0.92
Present	0.94	0.89	0.91
Macro Avg	0.92	0.92	0.92
Weighted Avg	0.92	0.92	0.92

Table 8.5: Fine-tuned BERT: **Accuracy=0.92**

Chapter 9

Conclusion

The source code for CyberGuard is available on github.
Deep learning techniques like LSTM and BERT perform better in classification of tweets based on sentiment: they can capture context efficiently while traditional models cannot. Leveraging BERT helped us understand how pre-trained models can be used for a wide range of different tasks.
We were able to achieve good metrics on dataset that had both English and Hinglish samples, thus we solved the problem in a more Indian context.

List of Figures

List of Tables

Bibliography

[1] S. Shahane, *Cyberbullying Dataset*, Kaggle. Available at: https://www.kaggle.com/datasets/saurabhshahane/cyberbullying-dataset.

[2] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 3rd ed., Prentice Hall, 2020. Available at: https://web.stanford.edu/~jurafsky/slp3/.

[3] "A review of cyberbullying detection techniques in social media," Available at: https://www.researchgate.net/publication/378875358_A_Systematic_Literature_Review_on_Cyberbullying_in_Social_Media_Taxonomy_Detection_Approaches_Datasets_and_Future_Research_Directions.

[4] Keras API Documentation. Available at: https://keras.io/api/.

[5] Scikit-Learn User Guide. Available at: https://scikit-learn.org/stable/user_guide.html.

[6] Karan Shah, Chaitaniya Phadtare, and Keval Rajpara, "Cyber-Bullying Detection in Hinglish Languages Using Machine Learning," *International Journal of Engineering Research & Technology (IJERT)*, vol. 11, no. 5, pp. 318–323, May 2022. Available: https://www.ijert.org/cyber-bullying-detection-in-hinglish-languages-using-machine-learning

[7] Manish Joshi, Dhirendra Pandey, Vandana Pandey, and Mohd Waris Khan, "A fusion framework for Hinglish cyberbullying detection using mBERT and FastText," *International Journal of Engineering in Computer Science*, vol. 7, no. 1, pp. 7–14, 2025. Available: https://www.computersciencejournals.com/ijecs/article/view/149/7-1-3

[8] Saurav Kumar, Mrinmoy Mondal, Tanuja Dutta, and Thoudam Doren Singh, "Cyberbullying detection in Hinglish comments from social media using machine learning techniques," *Multimedia Tools and Applications*, vol. 83, no. 36, pp. 84025–84046, Apr. 2024. Available: https://link.springer.com/article/10.1007/s11042-024-19031-z