Hate Speech Detection for Telugu Text

Gabriel Joshua, Varsini SR, Keerthana M, Karthick Roshan, Saptharishee M, Prathyuma V, Hareesh Teja S

This work studies various approaches to classifying hate speech in Telugu Text, using a dataset containing Telugu and English characters. Three pre-processing approaches were considered – Direct, Transliteration, and Transliteration. The pre-processed data form each approach was used independently to train ten models, broadly classified under three approaches – Machine Learning, Deep Learning, and Ensemble Learning. It was found that the transliteration pre-processing approach yielded better classification performance, with the single cell Bi-LSTM model most accurately performing the given classification task.

Keywords: Transliteration, Translation, Hate Speech, Tokenization, Ensemble Learning, Transformers

1 Introduction

In an increasingly interconnected and digital world, the rise of hateful speech in our communication media poses a significant challenge to maintaining respectful discourse in human interactions. Hateful speech, dubbed hate speech, characterized by expressions of hostility or discrimination against individuals or groups based on invariant attributes not only undermines the principles of freedom of expression, but has far-reaching implications on social harmony. As such, digital communication and media platforms have heavily invested in hate detection models to automate flagging and removal of hateful content in the attempt to curb said content.

This challenging task focuses specifically on hate speech detection in Telugu, a language spoken by people primarily from the Indian state of Andhra Pradesh and the Telangana. Detecting and addressing hate speech in Telugu is a rising concern given the language's growing adoption across digital media as the Indian population continues grow digital. The linguistic characteristics of Telugu, namely its unique script, vocabulary, and grammar necessitate research into this field to effectively filter hate speech.

The need for automated hate speech detection in Telugu is accentuated by the fact that a common approach for the same task for languages such as English are carried out via manual review. With more widely adopted languages this approach can be considered feasible as more of the global population is acquainted with the language, but manual detection for niche languages such as Telugu (in comparison to English) requires individuals who are fluent in reading and understanding Telegu, as well as any

underlying tones, and hence more expensive. By automating this process, not only is hate speech detection and moderate more feasible for digital platforms to employ, but the groundwork carried out in the development of work can be used for numerous other languages that share similar characteristics, as well as require similar approaches to process.

2 Literature Survey

In the ever-evolving field of natural language processing, a series of research papers have delved into detection of hate speech in multilingual contexts. [1] introduces a dataset comprising newspaper article headlines and labels to detect political bias in the articles with a novel model leveraging attention mechanisms which performs remarkably well, with potential applicability in other sentiment-based text classification domains as well. The proposed model involves a headline encoder, an article encoder, a headline attention layer, and a bias detection component. In terms of evaluation metrics, this study involves a negative log likelihood of correct labels as the training loss. The results, obtained through 5-fold cross-validation, demonstrate that the Headline Attention Network surpasses all other models, including traditional methods like Naive Bayes and LSTMs, achieving an accuracy of 89.54% in bias detection, notably superior to the best baseline accuracy of 85.25%.

[2] tackled the challenging task of recognizing dialog acts in code-mixed conversations, employing a dataset extracted from the Indian social media platform 'Chaibisket.' This dataset was meticulously annotated with six categories, encompassing English, Telugu, Mixed (morpheme-level mixing), acronyms, named-entities, and unknown words, reflecting the complexity of language interactions in the real world. The proposed methodology comprised a multi-stage pipeline, beginning with language identification, followed by data pre-processing, feature extraction, and ultimately, classification using a range of traditional algorithms that includes Support Vector Machines (SVM), K-Nearest Neighbors (KNN), Hidden Markov Models (HMM), Naive Bayes, Multi-Layer Perceptrons (MLP), and Long Short-Term Memory networks (LSTM). A sliding window-based splitting technique was adopted for sequence labeling in conversations which helped address the lack of annotated data. Their findings revealed that HMM excelled in language identification, boasting a remarkable F-score of 76.67, whereas LSTM emerged as the most effective algorithm in dialog act recognition achieving an impressive F-score of 70.5%.

The authors in [3] present a comprehensive study focused on part-of-speech (POS) tagging for code-mixed English-Telugu social media data. They addressed the unique challenges posed by code-mixed text, which combines elements from different languages. To do this, they collected a dataset of code-mixed English-Telugu data from Facebook, acknowledging the growing prevalence of such linguistic mixtures in social media and online platforms. The methodology involved the combination of POS taggers for individual languages, allowing them to effectively analyse and annotate code-

mixed text. Various classifiers, including Linear Support Vector Machines (SVMs), Conditional Random Fields (CRFs), and Multinomial Bayes, utilized different feature combinations to capture both the contextual and structural aspects of the text. They also explored the integration of monolingual POS taggers to enhance the accuracy of POS tagging for code-mixed data. It was noted that CRF performs better than SVMs and Bayes classifier. However, they noted some challenges, such as the absence of a normalizer for Telugu and the potential loss of contextual information when combining POS taggers from individual languages.

[4] focused on detecting hate speech and offensive content in multilingual code-mixed text, specifically in Malayalam and Tamil, using the HASOC-FIRE 2021 dataset. Traditional machine learning models uses character TF-IDF features, while neural networks employed deep learning techniques like Convolutional Neural Networks and Long Short-Term Memory Networks. The best performance was achieved by the MuRIL model, with a weighted F1-score of 0.636 for Tamil and 0.734 for Malayalam code-mixed data. Both BERT and MuRIL perform similarly on test data. Whereas [5] introduces a novel hate speech prediction model called Attentional Multi-Channel Convolution With Bidirectional LSTM Cell (AMC-BLSTM) that incorporates convolutional neural networks, bidirectional LSTM cells, and an attention mechanism to effectively capture local and global context in text data. The attention mechanism assigns importance scores to individual words, aiding the model in focusing on relevant terms for hate speech detection. Impressively, it outperformed five state-of-the-art and five baseline models on the imbalanced tweets datasets, achieving up to 91.5% accuracy and an F1 score of 0.91. However, the model's real-life effectiveness, particularly on imbalanced datasets suggests room for improvement through hyperparameter optimization and external knowledge integration, offering valuable insights for future research.

[6] employs a similar approach by deploying a range of machine learning models, with CNN+LSTM as the primary classifier. This architecture involves an embedding layer to convert words into vectors, a CNN layer for context understanding, an LSTM layer for sequential analysis, and a fully connected dense layer for the final output. During testing, the CNN+LSTM model outperformed existing methods with a remarkable 92.1% accuracy and training metrics of 0.92 for accuracy and F1 score.

[7] tackles the issue of hate speech detection in Bengali comments, leveraging a dataset of 7,425 comments. The methodology involves the application of classification algorithms like CNN, Bi-directional LSTM, GRU and a Recurrent Neural Network (RNN) with an Attention Mechanism for classification. The dataset is divided into binary and multi-class categories, with seven distinct labels with a novel addition of Bangla Emot module for understanding emotions conveyed through emojis and emoticons. The paper's evaluation showed the attention mechanism outperformed with 77% accuracy compared to other algorithms.

[8] investigates the problem of identifying hate speech in social media text using advanced multilingual Transformer models. The study utilizes the HASOC '20 dataset, containing tweets in English, German, and Hindi, which are annotated for a binary classification task distinguishing between hate speech and non-hate speech. The key innovation lies in the use of Transformer-based models, particularly XLM-RoBERTa (XLMR), for generating semantic embeddings for tweet texts, emojis, and segmented hashtags. These embeddings are then concatenated, creating a feature set that is fed through a two-layer multi-layer perceptron (MLP) for classification. Importantly, the authors adopt a unique approach to fine-tuning their model, employing learning rate scheduling based on macro F1-scores instead of traditional validation loss. This technique ensures that the model maximizes its performance during training. The results demonstrate the method's effectiveness, with high Macro F1 scores in hate speech detection for English, German, and Hindi, and strong fine-grained classification results for these languages.

Similarly for this approach of multilingual hate speech detection, [9] collected datasets from previous research in Hindi, English, and Indonesian languages. The collected data undergoes thorough pre-processing, and features are extracted using word and character n-grams, with SVM, NB, and RFDT classifiers evaluated through 10-fold cross-validation using the F1-Score as the primary metric. Results highlighted SVM with character quadgrams for Hindi, and SVM with word unigrams for English and Indonesian as the top-performing models. In the context of multilingual hate speech identification, the non-translated method proves most effective, although it presents challenges in terms of data and native annotators.

[10] focused on the Stormfront and GermEval datasets that have distinct annotation schemes, with Stormfront utilizing binary 'Hate' vs. 'noHate' labels, and GermEval employing a two-tiered schema with finer-grained labels. To address class distribution imbalances, oversampling was performed on the 'noHate' class of the Stormfront dataset, creating a nearly balanced distribution referred to as EN-OS for English training. Three hate speech classification models were introduced: a baseline linear SVM classifier, a CNN model, and a BiLSTM-CNN model. The fine-tuning on cross-lingual data enhanced classifier performance, but optimal performance was contingent on hyperparameter tuning. The average F1 scores of the models were relatively similar, both before and after fine-tuning, suggesting that more extensive training data might benefit neural models.

Table 1. Comparison of Datasets, Methodology and Metrics found in Research Works

Summary of Paper	Dataset	Best Proposed Methodology	Metric
[1]	Curated a dataset with Newspaper article headlines and labels	Headline Attention Network	Accuracy - 89.54%
[2]	English-Telugu conversational data was collected through WOz experiment	HMM LSTM	F-score of HMM 76.67, F-score of LSTM 70.5%.
[3]	English-Telugu data from Facebook	CRF	-
[4]	HASOC-FIRE 2021	MuRIL (Multilingual Representations for Indian Languages)	F1-score Tamil - 0.636 Malayalam - 0.734
[5]	Combined dataset from three Twitter sources	Attentional Multi-Channel Convolution With Bidirectional LSTM Cell (AMC-BLSTM)	Accuracy - 91.5% F1 score - 0.91
[6]	Hate tweets data - Kaggle	CNN+LSTM	Accuracy - 92.1%
[7]	Bengali comments from Facebook	Attention-based recurrent neural network	Accuracy - 77%
[8]	HASOC '20	Transformer model with XLM-RoBERTa (XLMR), for generating semantic embeddings for tweet	F1 score of Hindi dataset - 75.04 F1 score of English dataset - 90.2 F1 score of German dataset – 81.87
[9]	Dataset collected from Twitter API	SVM with character quadgrams for Hindi SVM with word unigram feature for English and Indonesian	F1 score of Hindi dataset - 66.03 F1 score of English dataset - 92.36 F1 score of Indonesian dataset 82.25
[10]	Stormfront and GermEval datasets	BiLstm	F1 score of German – 51.77 F1 score of English – 50.01

3 Methodology

Given that this research work is a comparative study, three pre-processing techniques were considered, along with three overarching modelling approaches.

3.1 Pre-processing

Under pre-processing, given that the dataset contained a mix of text using only Telugu characters, only English characters, and a combination of both alphabets, three approaches to pre-processing were considered – Direct Pre-processing, Transliteration, and Translation.

Each of the pre-processing techniques were independently applied to the training data and testing data provided.

Direct Pre-processing.
In this pre-processing technique, the data provided was initially cleaned by removing any non-alphanumeric characters and punctions, replacing multiple whitespaces with singular whitespaces, removing leading or trailing spaces, and finally replacing newline characters with whitespaces. Once cleaned, the text was parsed to classify the same based on the alphabet used in the text. Both tasks were performed using regular expressions.

Regular Expression.
Regular expressions, or regex, are a sequence of characters used to search for patterns in text, or pattern matching within strings. Regex allows for a set of rules, characters, and/or sequences to be set to define a search pattern that can be used to manipulate text and strings.

Tokenization using the Indic NLP Library.
Once the text was cleaned and classified based on alphabet type, the text was tokenized using the Indic NLP Library.

The Indic NLP Library is a Python library that provides tools for common text processing in Indian Languages, such as text normalization and word and sentence level tokenization. This was used to tokenize the text based on punctuation marks.

Finally, the cleaned text, the language of the text, and tokenized text were appended (column-wise) to the initial dataset to be used for model training later.

Transliteration.

Transliteration is the process of converting text or words from one alphabet to another. Unlike translation, transliteration focuses on representing the characters or symbols of one alphabet using the corresponding characters or symbols of another alphabet. The goal is to accurately represent the pronunciation of the original text in a different script.

Like the previous pre-processing approach, the data was initially cleaned by replacing multiple whitespaces with singular whitespaces using regex and removing any leading or trailing whitespaces.

AI4Bharat Indic-Transliteration.

Post cleaning, the AI4Bharat Indic-Transliteration Python library was used to transliterate any text that used the Telugu alphabet to the English alphabet. This is an AI-based transliteration engine that supports the 21 major languages used commonly in the Indian subcontinent. The transliterated text was appended to the initial dataset (column-wise) for further pre-processing.

Additional Cleaning.

Once the transliterated text was appended, it was further processed to remove any non-alphanumeric characters and punctions using regex, just as in the Direct Pre-processing approach. The text was also converted to lowercase.

Tokenization using the NLTK Library.

Since the text was transliterated to the English alphabet, a more standardized text and language processing library could be used to perform tokenization, such as the Natural Language Toolkit (NLTK).

Finally, the tokenized text was appended to the initial dataset as well to be used for model training.

Translation.

As the last pre-processing technique, the data was initially translated using the googletrans library, which internally uses the Google Translate Ajax API to return translated text.

Once translated, the text was cleaned and tokenized in the same manner as the Direct Pre-processing approach, i.e., using regex to clean the text by removing extra whitespaces and non-alphanumeric characters, and using the Indic NLP library to tokenize the text.

Finally, stopwords - common words that contribute little to the meaning of sentences, were removed as well using a reference list of stopwords from the NLTK

library, and regex was used to classify the text based on the alphabet used, similar to the Direct pre-processing approach yet again.

The initial translated text, tokenized text, cleaned text, cleaned text after removing stopwords, and the language (alphabet used) were appended to the initial dataset for model training.

3.2 Model Building

With the three sets of pre-processed data, 10 model architectures were built and trained to compare model performance, under three overarching branches – Machine Learning, Deep Learning, and Ensemble Learning.

Machine Learning.

Three machine learning classifiers were built for this study, Logistic Regression, Naïve-Bayes, and Support Vector Machines (SVM) classifiers.

Feature Extraction.

For the following three models, the Term Frequency-Inverse Document Frequency (TF-IDF) was calculated from the tokenized text from each of the pre-processing approaches. The TF-IDF is a statistic used to weigh the importance of words/terms found in a document.

The Term Frequency is defined by how frequently a term occurs in a document and is calculated as –

$$TF(t,d) = \frac{Number\ of\ times\ term\ t\ appears\ in\ document\ d}{Total\ number\ of\ terms\ in\ document\ d}$$

The Inverse Document Frequency is defined by the rarity of a term across a collection of documents and is calculated as –

$$IDF(t,D) = Log(\frac{Total\ number\ of\ documents\ in\ the\ corpus\ D}{Number\ of\ documents\ containing\ the\ term\ t})$$

The two statistics are multiplied to obtain the TF-IDF –

$$TF - IDF(t,d,D) = TF(t,d) * IDF(t,D)$$

The higher the TF-IDF statistic for a term in a document, the more important and distinctive the term is to that document relative to the entire corpus.

Logistic Regression.

A logistic regressor is a statistic-based classifier used to perform binary classification tasks (though it can also be extended for muti-categorical classification). The name of the algorithm is inherently a misnomer, as a logistic regressor, contrary to its name, is not used for regression tasks but instead for classification.

This algorithm applies a logistic (sigmoid) function to a linear combination of input features (i.e., a linear regression model) to essentially map the output of the linear regressor to a range of 0 to 1, which signifies the probability of belonging to the positive class.

The linear regression model is represented as –

$$y = w_1 * x_1 + w_2 * x_2 + \cdots + w_n * x_n + c$$

Where

- $x_1..x_n$ are the input features,
- $w_1..w_n$ are the weights assigned to each feature,
- c is the bias.

The sigmoid function used is described as such –

$$y = \frac{1}{1 + e^{-x}}$$

Where x in this case is the output of the linear regressor

Combined, the model for a logistic regressor is represented as –

$$P(y = 1) = \frac{1}{1 + e^{-(w_1 * x_1 + w_2 * x_2 + \cdots + w_n * x_n + c)}}$$

Where

- P(y = 1) is the probability of the instance belonging to the positive class.

Naïve-Bayes Classifier.

This is a probabilistic machine learning algorithm based in the Bayes' theorem, which describes the probability of an event based on prior knowledge of conditions that might be related to the event. The algorithm is "naïve" in that it assumes there is independence amongst the features, i.e., the features are weakly correlated.

For this study, a multinomial Naïve-Bayes classifier was considered. This is used commonly for text data as it is well-suited for handling data represented as vectors of

words counts or frequencies, such as the TF-IDF statistic calculated during feature extraction. Additionally, the multinomial distribution is more appropriate for modelling the probability distribution of the TF-IDF statistic as it is discrete data.

The probability model is derived from the multinomial distribution, and the classification decision is based on the maximum a posteriori (or MAP) estimation.

The model can be represented as such –

$$P(C_i|x_1, x_2 \dots x_n) \propto P(C_i) \prod_{j=1}^{n} P(x_j|C_i)$$

Where

- $P(C_i)$ is the prior probability of class C_i,
- $P(x_j|C_i)$ is the conditional probability of observing the word frequency x_j given that the document belongs to class C_i,
- $\propto$ denotes proportionality, given that the probability is typically proportional.

The classification decision is made by choosing the class that maximises the posterior probability and is given by –

$$\hat{y} = \arg\max_{C_i} P(C_i|x_1, x_2 \dots x_n)$$

Support Vector Machines (SVM).

SVM is a supervised machine learning algorithm used for classification as well as regression tasks. The primary objective of an SVM is to determine a hyperplane that best separates data into different classes.

SVMs are used for text classification tasks as they are deemed effective in handling high-dimensional data, such as representation of text as feature vectors, which in this case would be TF-IDF statistics.

The decision function of an SVM is defined by –

$$f(x) = sign(w \cdot x + b)$$

Where

- f(x) is the decision function that predicts the class label,
- w is the weight vector that defines the orientation of the hyperplane,
- x is the input feature vector (TF-IDF),
- b is the bias term

The optimization of an SVM model entails obtaining the optimal w and b that maximise the margin between the classes while minimizing classification error. The margin is the perpendicular distance from the hyperplane to the nearest data points from either class, known as support vectors.

This is formulated as –

Minimise

$$\frac{1}{2}\|w\|^2$$

Subject to constraints

$$y_i(w \cdot x_i + b) \geq 1 \; for \; all \; i$$

Where

- y_I is the class label (-1 to +1) of the i^{th} data point,
- x_i is the feature vector of the i^{th} data point.

Deep Learning.

Similar to the machine learning approach, four overall deep learning models were considered in this study, namely Long Short-Term Memory (or LSTM), Bi-directional Long Short-Term Memory (or Bi-LSTM) Bi-directional Gated Recurrent Units (or Bi-GRU) under sequential models, and transformers. For the sequential models, single cell and two-layer architectures of these models were studied.

Sequential models were considered due to their ability to capture and model sequential dependencies inherent in language. Text data is by nature sequential – the order of words and characters heavily alters the meaning of text. Sequential models can utilise this information to improve accuracy, in theory. Additionally, sequential models can capture long-range dependencies, which is required to understand the context and meaning of words that are spaced apart in a sentence but are correlated.

All the sequential model architectures contain an initial embedding layer that was used to convert the integer-encoded words into dense vectors of a fixed size, where the input dimension is set to the size of the vocabulary, the output dimension is set to 100, and input length is set to the maximum sequence length.

Long Short-Term Memory.

A Long Short-Term Memory (or LSTM) is a variant of a Recurrent Neural Network (RNN) architecture designed to capture long-range dependencies in sequential data.

The key feature of an LSTM in this application is this ability to store and retrieve information over long periods. This is achieved through a complex internal structure, in comparison to a traditional RNN, which incorporates a memory cell, input gate, output gate, and forget gate.

The memory cell is a separate component that stores information over time. This is what allows the architecture to capture and store long-term dependencies in sequential data.

The input gate is used to regulate the flow of information into the memory cell. This decides if the information from the current input should be stored in the memory cell.

The output gate is used to control the information that should be output from the memory cell to the next time step or to the final prediction, essentially filtering the information stored in the memory cell.

The forget gate is used to determine which information stored in the memory cell should be discarded. This enables the LSTM to "forget" less relevant information and rather focus on more recent inputs.

The single cell LSTM architecture for this application was designed as such –
Following the embedding layer is an LSTM layer with 64 neurons. The final output layer is a dense layer with a single neuron and a sigmoid activation function, and the Adam optimizer was chosen along with Binary Cross-Entropy as the loss function given that the architecture was designed to perform binary classification.

The two-layer LSTM architecture was designed as such –
Following the embedding layer is the first LSTM layer with 128 neurons. This is followed by another LSTM layer, this time with 64 neurons. This was done to enable the model to capture more intricate patterns in the data. There are three dense layers appended post the LSTM layers – The first dense layer has 64 neurons and uses the ReLU activation function. The second dense layer uses the ReLU activation function again, but with 32 neurons instead. The final layer output layer contains a single neuron and uses the sigmoid activation function. Like the single cell architecture, the Adam optimizer and binary cross-entropy loss function were used.

Bi-directional Long Short-Term Memory.

The fundamental reasoning for designing a Bi-directional Long Short-Term Memory (or Bi-LSTM) network remains the same as that of designing an LSTM network. The additional advantage that Bi-LSTM offers, in theory, is that it can capture contextual information from both past as well as future inputs and can hence capture dependencies from both sides of a given word or token.

The architectures for the single cell and two-layer Bi-LSTM networks remain the same as the LSTM networks discussed in the previous section but use a Bi-LSTM layer instead.

Bidirectional Gated Recurrent Unit.

A Bi-directional Gated Recurrent Unit (or Bi-GRU) is another variant of the RNN that incorporates gating mechanisms to control the flow of information within a cell.

Contrary to an LSTM, a GRU uses two gates, update and reset. The memory cell in a GRU is updated in a single step, unlike in an LSTM where what information must be updated is decided first, and then what information must be output is decided. In combination, these factors lead to the architecture of a GRU being simpler than an LSTM and with fewer parameters, making it computationally less expensive. This is traded for worse performance in capturing long-term dependencies as a GRU does not utilize a forget gate, which enables an LSTM to selectively forget certain information, instead of all stored information.

A Bi-GRU to a GRU is analogous to a Bi-LSTM to an LSTM in that the benefit offered is that a Bi-GRU processes data in both forward and reverse directions and can hence capture more context.

The architectures for the single cell and two-layer Bi-GRU networks remain the same as the LSTM networks discussed in the previous section but use a Bi-GRU layer instead.

Transformers.

A relatively young deep learning architecture, transformer models were introduced in the paper “Attention is All You Need” in 2017 (Vaswani et al.). Transformers have witnessed heavy adoption, becoming the de-facto models to be used for natural language processing tasks amongst others.

In general, transformers consist of three components, deviating it from traditional sequential models - a self-attention mechanism, positional encoding, and feedforward neural networks.

A self-attention mechanism is used to weight the importance of different words in a sequence based on contextual relationships, enabling the model to focus on relevant words when making predictions.

Positional encoding is used as transformers do not inherently capture the sequential order of words. Positional encoding is added to the input embeddings to provide information regarding the position of words in a sequence.

Feedforward networks are used for forward processing of the attended features, enabling the model to complex representations.

In combination, these attributes enable transformers to netter capture long-range dependencies in text, as well as process sequences in parallel, leading to more efficient training and better inferences.

Ensemble learning.

As the final approach, a stacking-based ensemble learning model was built. This technique seeks to combine multiple individual classifiers or base models to create a meta-classifier. The predictions of the base model serve as input features for the meta classifier, which makes the final prediction by aggregating the input features.

Random Forest, SVM, and Naïve-Bayes classifiers were chosen as the base classifiers and a Logistic Regressor as the meta classifier. The base classifiers were decided upon as they complement each other in terms of strengths and weaknesses and hence contribute to the overall robustness of the model.

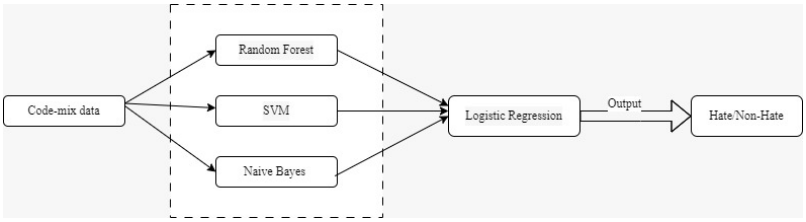

Fig. 1. Architecture of Stacked Ensemble Learning Model

4 Results

4.1 Metrics

To compare performance across the models, the following metrics were considered –

Accuracy.

Accuracy scores provide an overall measure of correctness in a classification task. Accuracy scores can tend to not be reliable for datasets with a class imbalance but still prove to be a decent singular metric to evaluate performance.

Accuracy is formulated as –

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}$$

Precision.

This is the ratio of correctly predicted positive observations to the total predicted positives, and essentially denotes accuracy of positive predictions.

Precision is formulated as –

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

Recall.

This is the ratio of correctly predicted positive observations to the total actual positives, measuring the model's ability to capture all positive instances.

Recall is formulated as –

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

F1 Score.

This is the harmonic mean of precision and recall, and is used to gauge the balance between both, considering both false positives and negatives.

F1 Score is formulated as –

$$Precision = \frac{2 * Precision * Recall}{Precision + Recall}$$

4.2 Model Comparison

Table 2. Comparison of Models and Metrics using Direct Pre-Processing

Model	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.7462	0.7432	0.7525	0.7478
Naïve-Bayes	0.7287	0.7886	0.6250	0.6973
SVM	0.7287	0.7886	0.6250	0.6973
Single Cell LSTM	0.6775	0.6408	0.8075	0.7146
Two-Layer LSTM	0.6612	0.6272	0.7950	0.7012
Single Cell Bi-LSTM	0.7325	0.7113	0.7825	0.7452
Two-Layer Bi-LSTM	0.7137	0.7422	0.6550	0.6958
Single Cell Bi-GRU	0.7162	0.7025	0.7500	0.7255
Two-Layer Bi-GRU	0.7350	0.7292	0.7475	0.7382

Stacked Ensemble Learning	0.7487	0.7571	0.7325	0.7445
Transformers	0.7175	0.7208	0.7100	0.7153

Table 3. Comparison of Models and Metrics using Transliteration Based Pre-Processing

Model	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.7575	0.7536	0.7650	0.7593
Naïve-Bayes	0.7375	0.7950	0.6400	0.7091
SVM	0.7375	0.7950	0.6400	0.7091
Single Cell LSTM	0.6812	0.6340	0.8575	0.7290
Two-Layer LSTM	0.7025	0.6832	0.7550	0.7173
Single Cell Bi-LSTM	0.7350	0.6807	0.8850	0.7695
Two-Layer Bi-LSTM	0.7387	0.7358	0.7450	0.7403
Single Cell Bi-GRU	0.7412	0.7188	0.7925	0.7538
Two-Layer Bi-GRU	0.7462	0.7408	0.7575	0.7490
Stacked Ensemble Learning	0.7575	0.7696	0.7350	0.7519
Transformers	0.7237	0.6916	0.8075	0.7450

Table 4. Comparison of Models and Metrics using Translation Based Pre-Processing

Model	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.7387	0.7369	0.7425	0.7397
Naïve-Bayes	0.7287	0.8080	0.6000	0.6886
SVM	0.7287	0.8080	0.6000	0.6886
Single Cell LSTM	0.6925	0.6509	0.8300	0.7296
Two-Layer LSTM	0.6975	0.6702	0.7775	0.7199
Single Cell Bi-LSTM	0.7212	0.6795	0.8375	0.7502
Two-Layer Bi-LSTM	0.7262	0.7139	0.7550	0.7339
Single Cell Bi-GRU	0.7137	0.6921	0.7700	0.7289
Two-Layer Bi-GRU	0.5987	0.5557	0.9850	0.7105
Stacked Ensemble Learning	0.7475	0.7591	0.7250	0.7416
Transformers	0.7075	0.6966	0.7350	0.7153

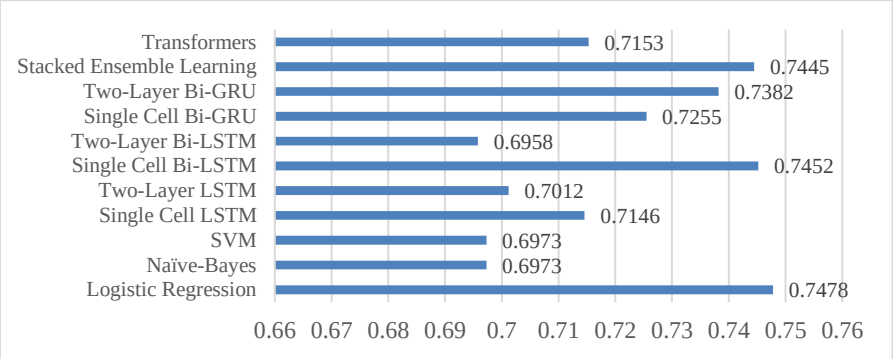

Fig. 2. Visualizing F1 Scores for All Models using Direct Pre-processing

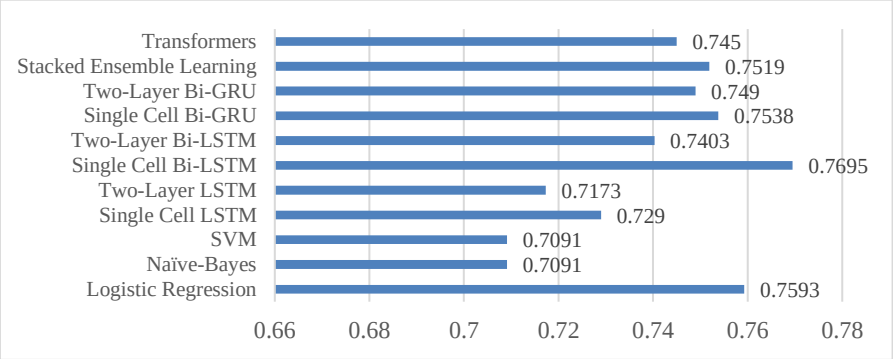

Fig. 3. Visualizing F1 Scores for All Models using Transliteration-Based Pre-processing

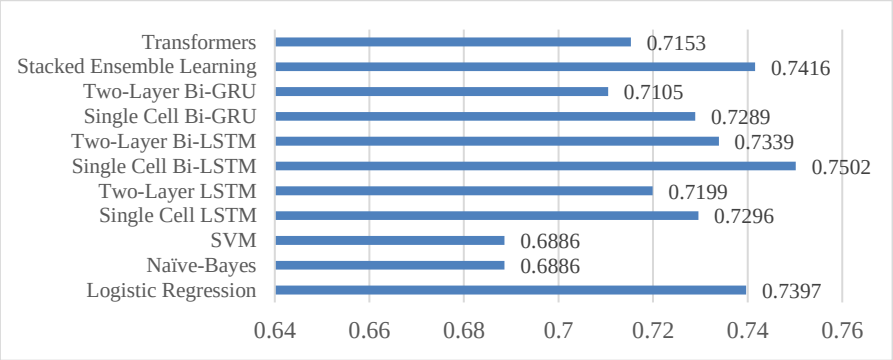

Fig. 4. Visualizing F1 Scores for All Models using Translation-Based Pre-processing

From the observed performance, it was deduced that single cell Bi-LSTM model using the transliteration pre-processing approach performed better for the given task, with an accuracy score of 0.7350 and F1 score of 0.7695.

5 Conclusion

Hence, a comprehensive study was conducted to perform Hate Speech in Telugu classification. Three pre-processing approaches were considered, along with ten model architectures and it was found that the transliteration approach performed better on average than direct and translation approaches, and that single cell Bi-LSTM model performed better at this classification task, indicating that it was advantageous to consider context of the text, in both forward and backward directions.

Link to Google Drive Folder

6 References

1. Gangula, R.R., Duggenpudi, S.R., & Mamidi, R. (2019). Detecting Political Bias in News Articles Using Headline Attention. BlackboxNLP@ACL.
2. D. S. Jitta, K. R. Chandu, H. Pamidipalli and R. Mamidi, "“nee intention enti?” towards dialog act recognition in code-mixed conversations," 2017 International Conference on Asian Language Processing (IALP), Singapore, 2017, pp. 243-246, doi: 10.1109/IALP.2017.8300589.
3. Nelakuditi, K., Jitta, D.S., Mamidi, R. (2018). Part-of-Speech Tagging for Code Mixed English-Telugu Social Media Data. In: Gelbukh, A. (eds) Computational Linguistics and Intelligent Text Processing. CICLing 2016. Lecture Notes in Computer Science(), vol 9623. Springer, Cham. https://doi.org/10.1007/978-3-319-75477-2_23
4. Bhawal, Snehaan & Roy, Pradeep & Kumar, Abhinav. (2022). Hate Speech and Offensive Language Identification on Multilingual code-mixed Text using BERT.
5. M. Fazil, S. Khan, B. M. Albahlal, R. M. Alotaibi, T. Siddiqui and M. A. Shah, "Attentional Multi-Channel Convolution With Bidirectional LSTM Cell Toward Hate Speech Prediction," in IEEE Access, vol. 11, pp. 16801-16811, 2023, doi: 10.1109/ACCESS.2023.3246388.
6. Bihari, Anand & Verma, Abhijeet & Singh, Anshuman & Tripathi, Sudhakar & Agrawal, Sanjay & Pandey, Shivendra & Verma, Sharad. (2023). Identification of Hate Speech on Social Media using LSTM. 17. 468-474.
7. Das, A., Al Asif, A., Paul, A. & Hossain, M. (2021). Bangla hate speech detection on social media using attention-based recurrent neural network. Journal of Intelligent Systems, 30(1), 578-591. https://doi.org/10.1515/jisys-2020-0060
8. Ghosh Roy, S., Narayan, U., Raha, T., Abid, Z., & Varma, V. (2021). Leveraging Multilingual Transformers for Hate Speech Detection. ArXiv, abs/2101.03207.
9. Ibrohim, M. O., & Budi, I. (2019). Translated vs non-translated method for multilingual hate speech identification in Twitter. International Journal on Advanced Science, Engineering and Information Technology, 9(4), 1116-1123. https://doi.org/10.18517/ijaseit.9.4.8123
10. Bigoulaeva, I., Hangya, V., & Fraser, A.M. (2021). Cross-Lingual Transfer Learning for Hate Speech Detection. LTEDI.