Content Moderation of Hinglish Comments

Mayank Nautiyal[1], Prarabdh Raipurkar[1], Pravesh Srivastava[1], Kavita Vaishnaw[2]
[1]Department of Electrical Engineering, [2]Department of Computer Science
{nautiyal_mayank, raipurkar_prarabdh, srivastava_pravesh, kavita.vaishnaw}@iitgn.ac.in
Indian Institute of Technology,Gandhinagar

Abstract

A recent study shows that 351.4 million Indians were active on various social media platforms in 2019 (Keelery, July 7 ,2020). With the ease of internet access, a large number of Indians actively participate in various social media discussions. Large population of India is bilingual and often expresses their views in Hindi-English code mixed data. Unlike English, hate speech detection in Hinglish is a challenge due to a lack of defined rules and structure of the sentences and the grammar. We have explored a variety of techniques from learning good word embeddings by fine-tuning the FastText model and using powerful sentence embedding techniques like concatenated power means. A profanity weighted sentence embedding scheme is also proposed which takes care of profane words and generate powerful embeddings. Various deep learning architectures like CNN1D, LSTM, BiLSTMs, HAN, and GRU Deep layer network are also utilized for comparison task.

1 Introduction

Many times debate in social media gets heated up at a personal level, and there are exchanges of hate comments. As a result, to maintain the decorum in social media content moderation has become the need of the hour. Even though these social networking sites are user-friendly, it becomes a perfect platform to put up their voice in front of the world. People aggressively use social media to express their views and opinions on various events, irrespective of the age, caste, creed, color and community. Unfortunately, the discussion often gets heated up, leading to the exchange of hate comments. People start putting reckless comments which leads to downfall of moral and rising revengeful thoughts in people. Targeted hate speech can disturb someone's mental well-being and cause mob riots, as witnessed in recent years. These problems makes the researchers to come up with solution and remove these hate speech contents from social media website which is a platform to discuss better ideas rather than spreading hate.

This problem is difficult to solve as code-switched language are often semantically complex and there is a lack of sophisticated methodologies (Chopra et al., 2020). The challenge becomes more demanding especially in country like India, because the population is diverse and there is huge bilingual and multilingual speaking population. Thus, the automatic code-mixed language faces huge challenge as there is no proper rules and high variations in grammar and spellings of the words (Sreelakshmi et al., 2020). (Mathur et al., 2018a) deals with this problem by providing Multi Input Multi Channel transfer learning model. Several LSTM and biLSTM model variations with custom word embeddings have been proposed to catch the long term sequences and abide to hate words in the sentences (Varade and Pathak, 2020). Some authors have also used convolutional networks to have a character level representations to solve this problem (Kamble and Joshi, 2018).

The remainder of this article is as follow: (i) Dataset; (ii) Modified HAN; (iii) Concatenated Power Mean Word Embeddings with FastText; (iv) Profanity Weighted Hate Speech Detection By Fine-tuning Pretrained Word Embeddings; (v) CNN, LSTM, BiLSTM and Deep Layer Network using GRUs; (vi) Conclusion.

2 Datasets

2.1 Hin-Eng Code Mixed Dataset

This dataset proposed by (Bohra et al., 2018) consists of 4575 code-mixed tweets. The annotation of the dataset is performed at two levels: 1.) Language at word level: In this, each word is associated with a tag of Hindi, English or another language; 2.) Hate or non-hate speech. This dataset has a *Kappa Score* of *0.982*, which proves the quality of the dataset.

2.2 IIITH Codemixed dataset

This dataset was proposed by (Prabhu et al., 2016a) and it contains 3879 comments taken from Facebook pages of Salman Khan and Narendra Modi .Dataset Contains: 15% Negative,50% Neutral and 35% Positive Comments. The comments were annotated by two annotators in a 3-level polarity scale - positive, negative or neutral. It has a Kappa score of 0.64 (Substantial interrater-reliability).

2.3 NITS-Hinglish-SentiMix SemEval-2020 dataset

This dataset is the part of SentiMix SemEval 2020 challenge (Patwa et al., 2020).It Contains 16995 instances of Hindi English Code Mixed data annotated on three levels – Negative, Positive and Neutral.It Contains 29.36% Negative, 37.61% Neutral and 33.03% Positive comments.

2.4 HEOT dataset

This dataset is taken from (Mathur et al., 2018b). It contains 2683 comments of Hinglish data which is annotated on three levels Non-Offensive,Abusive and Hate-inducing. For our purpose we have combined Abusive and Hate-inducing into a single class.

2.5 Examples

1. *"Mujhe apne manager se nafrat hai, I want to kill that guy."* – **Negative**
2. *"aur bhai kasise ho sab thik?".* - **Neutral**
3. *"app mujhe bahaut ache lagte ho"* - **Positive**

2.6 Data preprocessing

A function was created which can preform the following prepossessing operations:

1. **Noise removal**: All URLs, HTML tags, twitter handles (@), hashtags (#) were removed using python's regular expression library.
2. **Removing Punctuation marks and special symbols**: Another standard text processing technique involves removing of punctuation marks and special symbols since they generally don't add any value in text classification task.
3. **Text Normalization**: Lower casing is done to standardize all the tokens like "MODI","modi","Modi" to "modi" which is quite useful in text classfication task.
4. **Removing Stop words**: Stop words are commonly occurring words like 'a', 'the' ,'he', 'is' etc in the language.These stop words does not add any significance in sentiment classification tasks. NLTK Library have a list of all English stopwords. Hinglish stopwords were taken from the following Github repositray (Rana, 2013)
5. **Removing High/Low frequency words**: Ten most common and rare words were also removed during the pre-processing steps with the help of a hashmap.
6. **Removing Longer and shorter words**: All words of length less than 2 ("a") and greater than 12 were removed. Also words with digits were also removed like "abe123".

2.7 Problem with Hindi-English Code-mixed data

Hindi is phonetically typed, and words in Hindi can have diversity when written on the online platform, which leads to large amount of tokens. One such example is bahut which can be written as bahout, bohut, bhout, bauhat,bohot, bahut, bhaut, bahot etc.

Since there is no grammatical rules for Hinglish language, hence there are a lot of problemns like contractions of word like "between" "btwn", non-standard spellings such as "bhaiii" or "bhut bdiya" and non-grammatical constructions like "sir hlp plzz na".

3 Modified Hierarchical Attention Network

(Yang et al., 2016) introduced a very interesting model for document classification called Hierarchical Attention Network which exploits the hierarchical structure of the documents like "words makes sentences and sentences make documents". It uses attention mechanisms both at word and sentence level. (Prabhu et al., 2016b) introduced the subword level representations and showed that these representations serve as better linguistic units than character-level representations. So, we tried to incorporate this idea into HAN.But since here we are dealing with sentence classification task so me modified the HAN hierarchy to incorporate the idea that "sentences are made from words and words are composed of syllables".We modified the HAN by splitting the word into orthographic syllables and gave this syllable level decomposition as input to the embedding layer to get low-dimensional embeddings. These embeddings were fed to a syllable-level encoder and attention layer and further to a word-level encoder and attention layer. Bi-directional GRUs were used to capture the contextual information about the word.

We expected our modified HAN approach to work well in noisy text containing misspellings, common in Hinglish comments. Unfortunately, we were unable to achieve any improvement in the results.There were over-fitting issues with this model since the training accuracy was increasing but the validation loss was also increasing. We tried to mitigate this issue by adding L2 regularization,spatial dropout layers,early stopping but unfortunately we were not able to solve this problem. So, we decided to explore some more recent works such as concatenated pmeans, a hybrid approach for Hate Speech Detection by giving appropriate

weights to different Hinglish profane words and finally, we have shown commonly used Deep learning approaches to tackle this challenging task.

4 Concatenated Power Mean Word Embeddings with FastText

4.1 Power means

(Andreas Rücklé, 2018) introduced a powerful sentence embedding technique called power means embeddings. In p-means embedding we concatenates different word embeddings that represent different syntactic, semantic or sentiment information, resulting in a more diverse representation for the word embeddings.

In this approach instead of taking the standard average of the word embeddings it uses a more generalized representation called power means to perform different types of summarizations over word embedding dimensions.

Let $w_1, w_2, w_3.....w_n$ be word vectors of a sentence S then their power means can be calculated as follows:

$$\left[\frac{(w_1^p + w_2^p + ...w_n^p)}{n}\right]^{1/p}$$

For different values of p we can have different types of means:

1. For $p = 1$,it will reduce to the standard arithmetic mean.
2. For $p = 0$, we will have the geometric mean.
3. For $p = -1$, it will become the harmonic mean.
4. In the extreme cases, when p = $+\infty$ or - ∞, the power mean specializes to the minimum p = $-\infty$ and maximum p = $+\infty$ of the sequence.

These different types of means can later be concatenated to generate better sentence embeddings. One such concetenation could be $(+\infty \oplus 1 \oplus -\infty)$.

This method has shown significant imporvements over the various state-of-the-art supervised monolingual and cross-lingual adaptations like Infersent,SIF, Siamese-CBOW, and Sent2Vec according to (Andreas Rücklé, 2018)

4.2 Fine tuning Fasttext Model to generate word embeddings

To generate the word embeddings, we have fine tuned a supervised FastText Model on the Hinglish dataset. The model was fine-tuned on a pre-trained Fast text model previously trained on Wikipedia 2017, UMBC webbase corpus, and statmt.org news dataset and comprises of 16B tokens. Since we were fine-tuning on the pre-trained English model, hence a large labeled Hinglish dataset was required. For this purpose, we concatenated three different datasets (HEOT+SemEval+Hin-Eng).The resultant dataset have 24258 sentences with 46362 words.

Table 3 lists the hyperparameter used for training FastText model.

To check the quality of the learned word embeddings, multiple queries were made to the model one such query is:

model.get_nearest_neighbors('modi'):

(0.6394,'narendra'),(0.5979,'Modi')
(0.5768,'Mr.Modi'),(0.5638,'hindutva')
(0.5317,'bjp'),(0.5287,'Hindu-nationalist') (0.5262,'BJP'),
(0.5167,'modis')

This shows similar words have similar word vectors.

4.3 Concatenation

After learning the word embeddings, following combinations of power means were used to generate the corresponding sentence vectors.

1. $(+\infty \oplus 1 \oplus -\infty)$
2. $(1 \oplus 2 \oplus 3)$

First type uses arithmetic mean with extremums to generate the sentence embeddings.In second case arithmetic mean,quadratic mean and (mean with p = 2) were concatenated. Each word embedding have a dimension of 300 hence the concatenated p-means sentence embeddings will have a dimension of 900.

4.4 DOC2VEC

Doc2Vec is an extension of word2vec but unlike Word2Vec which simply converts a word into a vector, Doc2Vec not only does that but also combine them to form a sentence vector regardless of the sentence length. Since word ordering is maintained in Doc2Vec it can also capture the sentence semantics which is extremely useful in sentiment analysis task.
Hence for baseline purpose a CBOW DOC2VEC model was fine tuned on a large Wikipedia corpus with the hyperparameters given in Table 3.

4.5 Classification

Classification is done on IIITH dataset which is divided into two segments,(80%) training and (20%) testing.
Shape of feature matrix obtained from p-means embeddings is (N,900) where N is the number of training/testing sentences. Now for classification, different machine learning

FastText				
Model	**Concatenation**	**Accuracy**	**Recall**	**F1-Score**
SVM	$+\infty \oplus 1 \oplus -\infty$	66	59	61
SVM	$1 \oplus 2 \oplus 3$	68	60	63
RBF	$+\infty \oplus 1 \oplus -\infty$	65	56	59
RBF	$1 \oplus 2 \oplus 3$	63	55	57
XGBoost	$+\infty \oplus 1 \oplus -\infty$	67	60	62
XGBoost	$1 \oplus 2 \oplus 3$	67	61	63
Doc2Vec				
SVM	-	61	49	48

Table 1: Concatenated pmeans results

Hyperparameter	**Value**
Learning rate	0.008
Number of epoch	25
dim	300
loss	ova
lrUpdateRate	150000
Bucket Size	100000
maxn	5
minn	2
minCount	1

Table 2: FastText Training Hyperparameters

Hyperparameter	**Value**
dm	1
vector_size	300
min_count	1
window	5
negative	6
epochs	2
sample	1e-4

Table 3: DOC2VEC Training Hyper parameters

models were used. Following hyperparameters were obtained after applying the grid search on the following models.

1. **SVM**: *C = 1, gamma = 0.01, kernel = rbf*

2. **Random Forest Classifier**: *criterion= gini, max_depth = 5, max_features = auto, n_estimators= 500*

3. **XGBoost**: *learning_rate=0.1, max_depth=8, n_estimators= 140, objective=multi:softmax*

For these hyperparameters, results are shown in Table 1.

5 Profanity Weighted Hate Speech Detection By Fine-tuning Pretrained Word Embeddings

5.1 Fine-tuning pretrained Embeddings

(Al-Khatib and El-Beltagy, 2019) introduced a simple and effective approach for fine-tuning pretrained word embeddings for text classification tasks. We have tried pretrained Wikipedia2vec and FastText for generating 300-dimensional embeddings of words from the corpus. Fasttext embeddings are giving slightly more promising results for the Hinglish Hate Speech Detection task. Generally, label class is not used for generating pre-trained embeddings. In this paper, a class in which a specific word appears, acts as an additional contextual attribute during the fine-tuning process. Different words within a particular class will have vectors that are closer to each other in the space. Fine-tuning this way, we can generate robust final embedding vectors for all words in the corpus.

Along with general data-cleaning, we also remove around 180 English stopwords (e.g., my, been, had, etc.) and around 1000 Hinglish stopwords (e.g., isse, jab, jaise, abhi, waise, etc.) (Rana, 2013) from the corpus. We first represent our training data along with their label tags into the document format, and then we used in-built Doc2Vec modules of Gensim library for fine-tuning. Doc2Vec fine-tunes the pre-trained word embeddings as described in (Le and Mikolov, 2014). We kept the context window size of 4 and words embedding size of 300 length; fine-tuned for 100 epochs. Lastly, we save the embeddings in dictionary format.

5.2 Adding Profanity weights to slang words

Hinglish profanity list (Mathur, 2018) contains more than 200 slang words along with their profanity weights. Individual weights are on the scale of 1-10, where 10 (max) is assigned to the foulest slangs (e.g., mc, bc, etc.) and 1 (min) is assigned to normal words. Rest words are asigned weights of 1 value. This list is also converted to dictio-

nary format with words as keys and coressponding profanity weight as its value. Then, we use the following Algorithm 1 for generating sentence embeddings from fine-tuned pretrained word embeddings. Profanity weights play a major role in generating the sentences' embeddings based on how profane the words are in the sentence.

5.3 Classification of sentences

The dimensions of a sentence embeddings are kept (300,1) after fine-tuning word embeddings based on profanity. For the Hate Speech Detection task, we need to classify the sentences into three classes, i.e., Negative(0), Neutral(1) and Positive(2). We tried some Machine learning algorithms for this task, but better accuracies were achieved with Deep Learning techniques (CNN -1D model). In all experiments, we used the Keras library. The sentence embeddings are fed to the CNN model shown in the Figure 1. Our CNN model is very light, containing just four Conv1D layers with a Maxpool1D layer between 3rd and 4th layers. To avoid overfitting, we also apply a Dropout layer between 2nd and 3rd layers. We used the softmax activation function at the last dense layer. The hyper-parameters used in our model are shown in Table 4.

Algorithm 1 Generating Sentence Embeddings for the Dataset

Input: $Sen_1 \dots Sen_N$, Word Embeddings Dict (WED), Hinglish Profanity Dict (HPD)
Output: $Sentence\ Embeddings$ (300 dimensional)

$AllSenEmbd \leftarrow [\,]$
for $Sen \leftarrow 1$ to N **do**
  $SingleSenEmbd \leftarrow [\,]$
  for $words\ in\ Sen$ **do**
    if $word\ in\ WED$ **then**
      if $word\ in\ HPD$ **then**
        $SingleSenEmbd \leftarrow$
        $WED[word] * HPD[word]$
      else
        $SingleSenEmbd \leftarrow$
        $WED[word]$
      end if
    end if
  end for
  if $no\ single\ word\ of\ Sen\ in\ WED$ **then**
    $give\ random\ embedding\ to\ Sen$
  end if
  $AllSenEmbd \leftarrow SingleSenEmbd$
end for
return $AllSenEmbd$

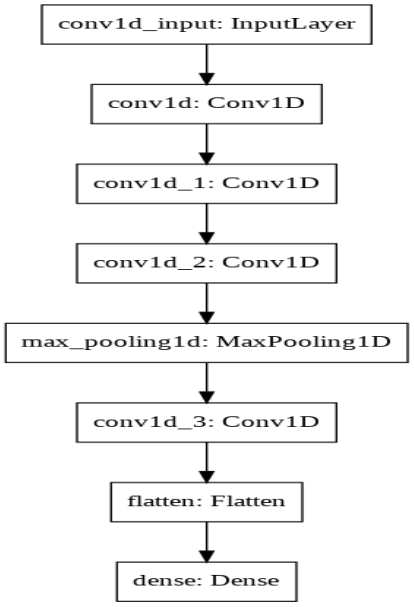

Figure 1: CNN model

Parameter Description	Values
Filter Size	4
Number of Filters	128,64,64,32,3
Activation function	Relu, Softmax
Optimizer	Adam
Learning Rate	0.001
Loss	Categorical CE
Epochs	100

Table 4: Hyper-parameters of CNN Model

5.4 Results on Various Datasets

Results are reported on three datasets with the same hyper-parameters mentioned in Table 4. Test-train split has been kept 0.2 in all the datasets. We have used only training data for fine-tuning the pretrained word embeddings and for training of the CNN model. Table 5 shows the performance of our approach in IIITH Code-mixed Dataset, NITS-Hinglish-SentiMix SemEval Dataset and HEOT Dataset respectively.

6 CNN, LSTM, BiLSTM and Deep Layer Network using GRUs.

In this section, we will describe four models in which first three models: (i) CNN 1D; (ii) LSTM; (iii) BiLSTM are inspired from (Kamble and Joshi, 2018) and the last model: (i) Deep Layer Network using GRUs is inspired from (Gupta, 2019) respectively. These models have shown superior performance for the classification of Hinglish comments in their original work. Formers have demonstrated

Quantitative evaluation of Profanity weighted approach				
Dataset	**Precision**	**Recall**	**F1-Score**	**Accuracy**
IIITH Codemixed Dataset	0.66	0.66	0.65	0.67
NITS-Hinglish SemEval Dataset	0.51	0.53	0.51	0.52
HEOT Dataset	0.82	0.83	0.82	0.83

Table 5: Performance on different datasets

their work for binary class classification, while the latter has been done for multi-class classification problems. An attempt is made to utilize their model and provide a comparative study with the P-Means model and Profanity weighted model to measure performance evaluation metrics' overall performance. In this section, we will look one by one at each model's architecture and hyperparameter values. In the end, performance metrics will be reported for these models.

6.1 Word Embeddings

To have the learning procedure better for these models, word embeddings have been generated using **gensim** library for training the Word2Vec model over Hinglish data. These embeddings have been learned by concatenating the available datasets to make a bigger dataset. In this way, an attempt is made to teach domain-specific embeddings for a bigger dataset. Even though the words are not limited to a specific dataset in social media platforms, the most common abusive hate word embeddings have been learned. The following specifications have been utilized for the embeddings:

- Embedding Size: 200, min-count = 1,iter = 10, sg = 1, hs =1

6.2 CNN 1D

The architecture of the model goes, as shown in Figure 2. An input layer takes the tokenized sentences of a particular vector length as an input to the model. It is then passed to the embedding layer, where the weight matrix is used from the trained embeddings. The trainable parameter for the embedding layer is kept False here. The third layer is the three filters to perform 1d convolution is used. After this, a global maxpooling layer is utilized. After passing through the dropout layer, this layer's output is concatenated together and is given to the dropout layer, followed by a dense layer.

Following specifications have been utilized for the model:

- Embedding Dimension = 200.
- Number of filters =3, Size of filters = 64, Dropout = 0.5
- Pooling layer= GlobalMaxPooling

Figure 2: CNN 1D

- Dense layer units =3, activation function = softmax, loss = categorical crossentropy loss, optimizer = Adam.

Table 6 reports the performance metrics for the CNN-1D model. The precision, recall, and F1 score is reported for each class along with training and testing accuracy of the model on the dataset. The testing dataset consisted of 2000 sentences and was kept aside from training and validation data for the model's performance evaluation. From the results shown, we can say that the overall performance of the model is decent. Even though the accuracy doesn't reach heights, however a decent F1 score is witnessed. The testing accuracy reported is 62 %. This seems that the performance is not bad considering three classes are written. The

Table 6: Performance of CNN 1D Model

Class	Precision	Recall	F1-Score
0	0.64	0.62	0.63
1	0.57	0.59	0.58
2	0.67	0.66	0.66
Train Accuracy= 0.86		**Test Accuracy=0.62**	

previous works have reported 82% accuracy for binary class classification for this model.

6.3 LSTM

The architecture of the model is shown in figure 3. The model starts with the Input layer, where it takes the tokenized vectors as input and is followed by the embedding layer. The embedding layer has the weight matrix that has been created using the trained Word2Vec model using the gensim library on the mentioned datasets. The trainable parameter is kept False here also. The LSTM layer follows this layer. After this, the GlobalMaxPooling1D layer is added. The output of this layer is passed through the Dense layer, and the final classification is done.

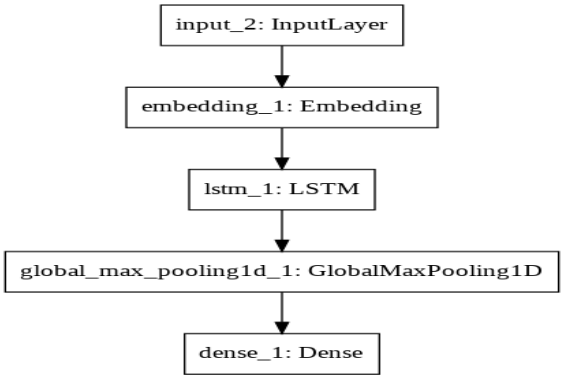

Figure 3: LSTM Model

The specifications utilized for this model are:

- Embedding Dimension = 200.
- Number of LSTM units =100, Recurrent Activation = "Hard Sigmoid", recurrent dropout = 0.2
- Pooling layer= GlobalMaxPooling
- Dense layer Units =3, Activation = softmax, Loss = Categorical Crossentropy, Optimizer = Adam.

Table 7: Performance of LSTM model

Class	Precision	Recall	F1-Score
0	0.58	0.66	0.62
1	0.55	0.57	0.56
2	0.67	0.56	0.61
Train Accuracy= 0.91		**Test Accuracy= 0.591**	

Table 7 reports the performance of the LSTM model keeping into account the value of different metrics for each class in the dataset. It appears that from the values, the performance of this model lies close to the performance of the CNN 1D model. A closer look tells that the performance of CNN 1D is slightly superior to the LSTM model. The testing accuracy reported is around 59.1% with an F1 score of 0.62 for one of the class. This also shows that the model's performance is not too attractive; however, it is decent compared to the works presented by other authors for the classification of Hinglish data.

6.4 BiLSTM

The architecture of the model is shown in Figure 4. The model takes in input as its first layer, which is followed by the embedding layer. Once again, the embedding layer holds the weight matrix, which is obtained by training the Word2Vec model on the available Hinglish dataset utilizing the gensim library. The trainable parameter is kept False here also. From the embedding layer, the output is passed to the Bidirectional LSTM layer. GlobalMaxPooling1D layer is added after the biLSTm layer. Finally, a Dense layer is added to generate the predictions for the final classification task.

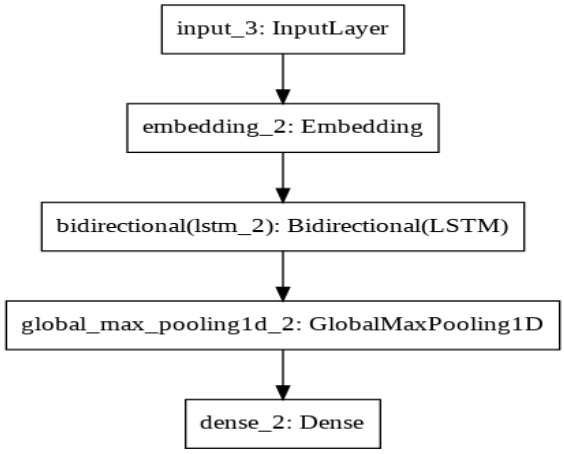

Figure 4: biLSTM Model

The specifications utilized for this model are:

- Embedding Dimension = 200.
- Number of LSTM units =100, Recurrent Activation = "Hard Sigmoid", recurrent dropout = 0.2
- Pooling layer= GlobalMaxPooling
- Dense layer Units =3, Activation = softmax, Loss = Categorical Crossentropy, Optimizer = Adam.

Table 8 reports the biLSTM model's performance, keeping into the consideration of values of different metrics for the performance evaluation of the model. A closer look at the tabulated result of the biLSTM model shows that its

performance is similar to the LSTM model in terms of accuracy. However, the precision, recall and F1 score value offers a very slight superior performance over the LSTM model. Overall it also lies in the proximity of the version of CNN 1D and LSTM model.

Table 8: Performance of BiLSTM Model

Class	Precision	Recall	F1-Score
0	0.62	0.58	0.60
1	0.54	0.58	0.56
2	0.64	0.62	0.63
Train Accuracy= 0.958		**Test Accuracy= 0.591**	

6.5 Deep Layer Network using GRU

The network architecture presented in this section is inspired by (Gupta, 2019). It has multiple GRU layers followed by the Dense and maxpooling layer. The deep network is chosen with multiple GRU layers to capture the long term dependencies in the sentences. A deeper network tends to learn better weights and thus provide better performance over other simple networks. The original work (Gupta, 2019) for this model reported 92% Precision value and Recall rate of 88%. An attempt is made to reproduce or outperform the scores presented.

The model's architecture is as follows: The model starts with an input layer where it is fed with a tokenized word vector of the sentences. This is followed by a similar embedding layer, where the embedding matrix is obtained from the Word2Vec model trained by the gensim library on the available Hinglish dataset. The trainable parameter is kept False here also. After the embedding layer follows four bidirectional GRU layer consecutively. After this GRU layer, a Dense layer is added, following which comes the dropout layer. Again a Dense layer is added. The penultimate layer which follows this Dense layer is the GlobalMaxPooling layer. The highest layer is also a Dense layer, after which the final classification results are obtained. This architecture makes the network much more profound, and with the presence of the GRU layer, long term dependencies can be obtained.

An attempt was also made with the Bidirectional LSTM layer instead of the GRU layer; however, due to GRU's simple structure and a little speeded up performance, GRU was found to be a suitable choice for this architecture.

The specifications utilized for this model are:

- Embedding Dimension = 200.
- Number of GRU units =64, Recurrent Activation = "Sigmoid", recurrent dropout = 0.2
- Pooling layer= GlobalMaxPooling

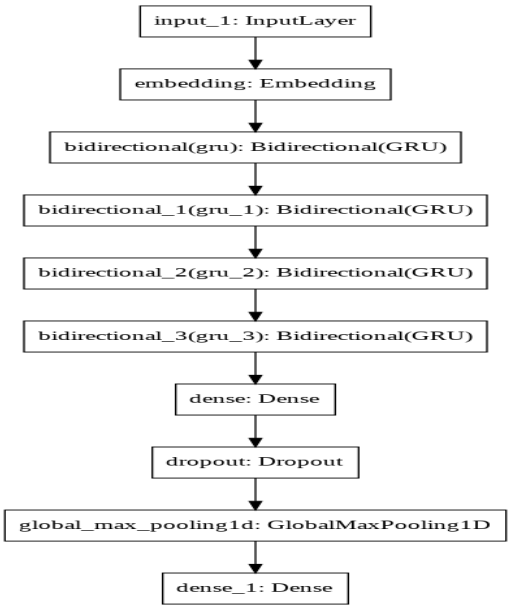

Figure 5: Deep Layer Network using GRUs

- Dense layer Units =3, Activation = softmax, Loss = Categorical Crossentropy, Optimizer = Adam, learning rate = 0.001.
- Dropout layer 0.2

Table 9 reports performance metrics of this model. The table shows it has better performance than the model mentioned above in the comparative study. The testing accuracy for this model has been reported as 60.2% and the highest F1 score obtained is to be 0.61 for one of the classes. The Precision value also shows an appreciable rise of 0.69. Overall the performance in comparison is a little improved. Even though the performance is superior, the model was unable to perform as per the performance reports mentioned in this model's original work. This lower performance points to the fact that: (i) The custom modifications made didn't prove to be sufficient to boost the model's performance. (ii) The embeddings trained can be further improved to enhance the model's performance.

Table 9: Performance of Deep Layer model using GRUs

Class	Precision	Recall	F1-Score
0	0.61	0.61	0.61
1	0.55	0.63	0.59
2	0.69	0.56	0.62
Train Accuracy= 0.797		**Test Accuracy= 0.602**	

6.6 Implementation and Performance Comparison

An attempt was made to reproduce the results for previous work for these models and utilize the models for comparing the overall performance of the P-means model and Profanity Weighted model. However, the results are not up to the expectation, even though the Precision, Recall, and F1-score indicate slightly better performance. The overall expectations with these models are much higher than those reported in this article.

The lower performance score of these models can be attributed to the following possible reasons:(i) The custom embeddings trained are not up to the mark. They can be made better by utilizing other methods such as GloVe, fasttext, and other custom embedding models. (ii) The optimum choice of hyperparameters made here is not made with the consecutive loop and grid search; instead, the values are inspired by the previous works after making substantial hit and trial attempts.

Each of the models has been trained for 20 epochs with experimentation with hyperparameters. Batch-size has been chosen to be between 16 and 64 (both inclusive) for the mentioned models. The optimum number of epochs has been selected after examining the cross-validation and training loss of the models.

Conclusions

Due to incredible growth in internet users, Indians have become comfortable with the Hinglish language for text conversations. Learning good word embeddings is important but to generate good sentence vectors from them is equally important. Fine-tuning a Fasttext model on the large Hinglish dataset can generate good word embeddings, and by using concatenated pmeans, we can generate good sentence vectors.

In Hinglish, there are some profane words which are most commonly used over the internet. While generating sentence embeddings, we can take care of the presence of these words according to their profanity level. This profanity weighted approach helps to generate powerful sentence embeddings for hate detection task specifically in Hinglish language.

In section 6, an attempt is made to get the hold of performance comparison of various in practice architecture for hate speech detection. Although the results shows decent performance on training and testing data, the results for these models were little demotivating when deployed as web application. The predictions for these models comes out for detecting every word present as hate speech or other classes. This although provides a scope for profounding future work, however the results may not be promising enough.

Looking at the flow of work mentioned in this article, it can be divided into two groups: (i) The first group of models tries to capture the task with more inclination towards the generation and fine tuning the embeddings. (ii) The second group of models tries to bring out the various models which can capture the long term dependencies based on embeddings generated from simple Word2Vec models. This leaves a hope for future work where one can fuse these two techniques and can claim better performance of hate speech detection.

References

Amr Al-Khatib and Samhaa R. El-Beltagy. 2019. A simple and effective approach for fine tuning pre-trained word embeddings for improved text classification.

Maxime Peyrard Iryna Gurevych Andreas Rücklé, Steffen Eger. 2018. Concatenated power mean word embeddings as universal cross-lingual sentence representations. *https://arxiv.org/abs/1803.01400*.

Aditya Bohra, Deepanshu Vijay, Vinay Singh, Syed Sarfaraz Akhtar, and Manish Shrivastava. 2018. A dataset of Hindi-English code-mixed social media text for hate speech detection. In *Proceedings of the Second Workshop on Computational Modeling of People's Opinions, Personality, and Emotions in Social Media*, pages 36–41, New Orleans, Louisiana, USA. Association for Computational Linguistics.

Shivang Chopra, Ramit Sawhney, Puneet Mathur, and Rajiv Ratn Shah. 2020. Hindi-english hate speech detection: Author profiling, debiasing, and practical perspectives. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 386–393.

Vivek Kumar Gupta. 2019. " hinglish" language–modeling a messy code-mixed language. *arXiv preprint arXiv:1912.13109*.

Satyajit Kamble and Aditya Joshi. 2018. Hate speech detection from code-mixed hindi-english tweets using deep learning models. *arXiv preprint arXiv:1811.05145*.

Sandhya Keelery. July 7 ,2020. Number of social network users in india from 2015 to 2018 with a forecast until 2023. *arXiv preprint arXiv:1810.04805*.

Quoc V. Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents.

P Mathur. 2018. Hinglish profanity list. https://github.com/pmathur5k10/Hinglish-Offensive-Text-Classification.

Puneet Mathur, Ramit Sawhney, Meghna Ayyar, and Rajiv Shah. 2018a. Did you offend me? classification of offensive tweets in hinglish language. In *Proceedings of the 2nd Workshop on Abusive Language Online (ALW2)*, pages 138–148.

Puneet Mathur, Ramit Sawhney, Meghna Ayyar, and Rajiv Shah. 2018b. Did you offend me? classification of offensive tweets in Hinglish language. In *Proceedings of the 2nd Workshop on Abusive Language Online (ALW2)*, pages 138–148, Brussels, Belgium. Association for Computational Linguistics.

Parth Patwa, Gustavo Aguilar, Sudipta Kar, Suraj Pandey, Srinivas PYKL, Björn Gambäck, Tanmoy Chakraborty, Thamar Solorio, and Amitava Das. 2020. Semeval-2020 task 9: Overview of sentiment analysis of code-mixed tweets. In *Proceedings of the 14th International Workshop on Semantic Evaluation (SemEval-2020)*, Barcelona, Spain. Association for Computational Linguistics.

Ameya Prabhu, Aditya Joshi, Manish Shrivastava, and Vasudeva Varma. 2016a. Towards sub-word level compositions for sentiment analysis of hindi-english code mixed text. *CoRR*, abs/1611.00472.

Ameya Prabhu, Aditya Joshi, Manish Shrivastava, and Vasudeva Varma. 2016b. Towards sub-word level compositions for sentiment analysis of hindi-english code mixed text.

Shivam Rana. 2013. Hinglishnlp. https://github.com/TrigonaMinima/HinglishNLP/tree/master/data/assets.

K Sreelakshmi, B Premjith, and KP Soman. 2020. Detection of hate speech text in hindi-english code-mixed data. *Procedia Computer Science*, 171:737–744.

Rahul S Varade and Vikas B Pathak. 2020. Detection of hate speech in hinglish language. In *Machine Learning and Information Processing*, pages 265–276. Springer.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, San Diego, California. Association for Computational Linguistics.